Année : 2004
No attribué par la bibliothèque

|_|_|_|_|_|_|_|_|_|_|

THÈSE

présentée à
l'**ÉCOLE POLYTECHNIQUE**

pour obtenir le titre de

DOCTEUR EN SCIENCES

Spécialité
Informatique

soutenue par

Didier LE BOTLAN

le 06/05/2004

Titre

ML$^{\mathsf{F}}$: Une extension de ML avec polymorphisme de second ordre et instanciation implicite.

Directeur de thèse : Didier Rémy
INRIA Rocquencourt

Jury

Directeur	Didier Rémy
Rapporteurs	Benjamin Pierce Jacques Garrigue
Président	Roberto Di Cosmo
Examinateurs	Claude Kirchner Dale Miller

Abstract

We propose a type system ML$^{\mathrm{F}}$ that generalizes ML with first-class polymorphism as in System F. Expressions may contain second-order type annotations. Every typable expression admits a principal type, which however depends on type annotations. Principal types capture all other types that can be obtained by implicit type instantiation and they can be inferred.

All expressions of ML are well-typed without any annotations. All expressions of System F can be mechanically encoded into ML$^{\mathrm{F}}$ by dropping all type abstractions and type applications, and injecting type annotations of lambda-abstractions into ML$^{\mathrm{F}}$ types. Moreover, only parameters of lambda-abstractions that are used polymorphically need to remain annotated.

Keywords:

Type Inference, First-Class Polymorphism, Second-Order Polymorphism, System F, ML, Type Annotations.

Résumé

Nous nous intéressons à une extension de ML avec polymorphisme de première classe, à la manière du Système F. Cette extension, nommée ML$^{\mathrm{F}}$, utilise les annotations de types d'ordre supérieur données explicitement dans le programme pour inférer de manière principale le type le plus général. Toute expression admet ainsi un type principal, qui dépend des annotations présentes initialement dans le programme.

Toute expression de ML est typable dans ML$^{\mathrm{F}}$ sans annotation supplémentaire. Les expressions du Système F sont encodées de manière systématique dans ML$^{\mathrm{F}}$ en supprimant les abstractions et les applications de types, et en traduisant les annotations de types dans le langage de types de ML$^{\mathrm{F}}$. De plus, les paramètres de lambda-abstractions qui ne sont pas utilisés de manière polymorphe n'ont pas besoin d'être annotés.

Mots Clefs

Inférence de types, Polymorphisme de première classe, Polymorphisme d'ordre deux, Système F, ML, Annotations de type

Contents

Acknowledgments

Ma propre expérience, quoique limitée, m'incite à croire que mener à bien une thèse reste moins difficile qu'élever un bébé. Cependant, ceci ne révèle rien de la difficulté à *diriger* une thèse. Je tiens à remercier chaleureusement Didier Rémy pour avoir accepté le rôle de directeur de thèse qu'il a exercé avec une attention et une patience exemplaires. Parmi une multitude d'autres enseignements, il m'a montré l'importance de prendre du recul en recadrant le travail en cours, même inachevé, dans un contexte plus large. J'ai pu bénéficier avec joie de sa connaissance pointue de TEX , incarnée notamment dans le très utile WhizzyTEX .

Je remercie tout particulièrement Benjamin Pierce et Jacques Garrigue, rapporteurs, pour l'attention particulière qu'ils ont portée à mon travail, ainsi que les nombreuses heures de décalage horaire qu'ils ont dû subir pour participer à la soutenance. Leurs suggestions furent particulièrement bienvenues et me seront profitables à l'avenir. Merci également aux autres membres du jury pour l'intérêt qu'ils ont manifesté envers ma thèse : Roberto Di Cosmo, Claude Kirchner, et Dale Miller. Je remercie au passage Roberto pour son intervention salvatrice sur mon ordinateur portable, réalisée grâce à Demolinux.

Merci à tous les membres de ma famille pour leur soutien inépuisable : mes parents, mes beaux-parents, mon épouse, et à sa manière, mais tout aussi inépuisable, Pierre-Gilles.

Merci à Vincent Simonet pour ses multiples compétences, notamment sa relecture minutieuse et sa maîtrise de TeX. J'ai aussi apprécié l'aide linguistique apportée par James Leifer.

Tous les membres du projet Cristal, passés et présents, se caractérisent par une disponibilité exceptionnelle. Merci en particulier à Xavier Leroy qui prend toujours le temps de partager ses vastes connaissances alors même que sa nouvelle fonction de directeur de projet ne lui laisse que peu de temps.

Je remercie Jean-Jacques Lévy pour avoir su présenter la recherche en informatique de manière motivante, ainsi que pour les sorties piscine. Maxence Guesdon étant complice, je le remercie également.

Introduction

In 1965, Gordon Moore made an observation which has since become famous: the number of transistors per integrated circuit grows exponentially. This singular phenomenon has been maintained for forty years. As a consequence, computers are more and more powerful and fast, allowing more and more programmers to write ever larger and more complex programs. A less pleasant consequence is that the number of ill-written programs follows the same curve. Fortunately, research on programming languages brings solutions to help the programmer in keeping his mental health. One of the most efficient technique is *static typechecking*. Static typechecking can be viewed as a validation done by the compiler that guarantees that the program will not perform illegal operations, that is, will not crash. For instance, it ensures that every data structure will be manipulated only by suitable functions. We usually say that "well-typed programs do not go wrong". Moreover, in practice, a large proportion of errors is automatically detected by typechecking, and this considerably accelerates program development.

In order to typecheck a program, a compiler may require explicit type information in the source program itself, as is the case in C or Java, where each variable and function declaration needs to be annotated with types. This is acceptable when the types are simple. However, in order to get more expressiveness power and to be able to typecheck more complex programs, it is necessary to introduce a richer syntax for types. This is the case for example with higher-order functional languages, such as ML [DM82]. Then smart compilers perform *type inference* (also called *type reconstruction*) in order to find by themselves the type annotations. In the case of ML, the type inference algorithm is complete, which means that no type annotation is needed in the program, and all the information may be inferred by the typechecker (*i.e.* by the compiler). It is still possible, though, to write explicit type annotations for documentation purposes.

Types provide an approximation of programs and the typechecker may reject well-behaved programs it cannot type. Such a limitation can be encountered in monomorphically typed systems, where each function can only be used with one single type. In that case, a given function cannot be applied to two data structures with incompatible types. For example, the function `List.length`, which returns the number of elements of the list it is applied to, could only be used with lists of integers, and another function should be defined to operate on lists of strings. However, both functions have the same code, but only their types are different. This useless duplication of code can be avoided thanks to more expressive type systems. Indeed, many solutions have been proposed in order to typecheck functions applied to arguments of incompatible types. They include type systems based on *subtyping*, *intersection types*, and *parametric polymorphism*. We immediately discard the solution used in C or Java which consists in casting the type of expressions to `void*` or `Object` in order to put them in a general-purpose structure. When taking an element out of the structure, some unsafe operation, such as a cast from `Object` to a given class or from `void*` to a given pointer type, is needed. In the worst case it breaks type safety and well-typed programs do not necessarily behave well; in the best case dynamic typechecks are performed at run-time. In both cases, some type errors are only detected at run-time, which is not satisfying in our opinion. We describe quickly the other solutions mentioned above.

Subtyping allows some information in types [Car88, Mit83, Pot98b] to be forgotten. For example, integers and floating point numbers are usually of incomparable types, but both can be viewed as members of the more general type "number". A list of integers can then be viewed as a list of numbers, or even as an unspecified list, forgetting the type information about the content of the list. To pursue the example given above, the function `List.length` can then be applied to any unspecified list, including, by subtyping, lists of integers and lists of strings. Extensions of ML with subtyping that allow for type inference have been designed [OSW99, Pot98a]. They remain to be tested on large-scale developments. One step towards integration within a large-scale framework is Flow Caml [Sim03], an information flow analyzer for the Caml language based on subtyping.

Intersection types enumerate precisely the different types a given value can have, and may be viewed as a refinement of subtyping [CDC78, CDCS79, Sal82, Pot80, Pie91]. For instance, `List.length` can be given an intersection type that states that it is both a function accepting lists of integers *and* a function accepting lists of strings. Intersection types are in some way as expressive as programs themselves [CC90]. In particular, a program is typable with intersection types if and only if it terminates [Pot80]. Some restrictions based on ranks have been proposed to integrate intersection types within a programming language with type inference [KW99, Jim00]. However, such restrictions interfere with modularity: a value cannot always be passed as an argument if the rank of its type is too high. Indeed, the type of the function being applied needs to be of a higher rank, which might be forbidden. Hence, although intersection types enjoy useful properties, they are too expressive and must be limited in practice.

Parametric polymorphism, which we simply call *polymorphism*, allows quantification over some *type variables* in types. Then any polymorphic expression may have an infinite number of types. For example, `List.length` is a function accepting lists of α, whatever α is. We develop our presentation of parametric polymorphism hereafter.

These solutions provide a way to use a single function with different types, that is, to apply a single function to data structures with incompatible types. They all help to avoid unnecessary duplication of code, and therefore contribute to keeping the program concise. In the ML language, the third solution, parametric polymorphism, is used. More precisely, the ML typechecker is able to infer polymorphic types for named functions. In particular all functions defined at top-level are assigned a polymorphic type if possible. This feature greatly contributes to ML's conciseness and expressiveness. Moreover, even though the cost for type inference is exponential in theory, it appears to be very efficient in practice. The success of ML suggests that parametric polymorphism is an expressive feature that applies well to large scale programs.

Second class polymorphism

Polymorphism in ML is second-class only: as mentioned above, only named values may have a polymorphic type, that is, only values bound with a `let` construct. As a consequence, it is not possible for the argument of a function to be used with a polymorphic type. For instance, let `id` be the identity function, that is, $\lambda(x)\ x$. Its type in ML is polymorphic and equal to $\forall\,(\alpha)\ \alpha \rightarrow \alpha$, which we write $\sigma_{\texttt{id}}$. Then $(\lambda(x)\ x\ x)$ `id` **($\underline{1}$)** is not typable in ML, whereas `let` x = `id in` $x\ x$ **($\underline{2}$)** is. Indeed, typing the body of the function $\lambda(x)\ x\ x$ would require x to have a polymorphic type, *e.g.* $\sigma_{\texttt{id}}$. However, x is not bound by a `let` but by a λ. Hence, its type is monomorphic in ML, and the typing of (1) fails. On the contrary, x is first bound by a `let` construct in (2), and then the body $x\ x$ is typed. In ML, this allows us to infer a polymorphic type for x, namely $\sigma_{\texttt{id}}$, which suffices to typecheck (2). This example illustrates why polymorphism in ML is second-class only; that is, because it is not allowed in λ-abstractions. It also shows that the `let` construct may, in some situations, circumvent this limitation. However, there are some situations where `let`-polymorphism *a la* ML is not enough.

Motivation

We detail our motivation for this thesis whose contribution is a programming language named ML^{F}. Our starting point is ML.

Actually, ML is surprisingly powerful in defining, handling, and iterating over complex data structures. Indeed, data structures are usually typed using parametric type constructors such as built-in types or user-defined datatypes. The type parameter represents the type of values to be found in the structure. As an example, a list of elements with type α is given the parametric type α `list` in OCaml [LDG$^+$02]. Then the associated constructors such as the empty list, the cons operator, or user-defined constructors are automatically given a polymorphic type with outer quantification. For example, the empty list is given the type $\forall\,(\alpha)\ \alpha\ \texttt{list}$. Most functions defined on these data structures, such as iterators, are also given polymorphic types with outer quantification. For instance, `List.fold` is given the polymorphic type $\forall\,(\alpha,\beta)\ \beta\ \texttt{list} \rightarrow (\alpha \rightarrow \beta \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha$.

As long as the data structure is known to the programmer, ML is perfectly suitable, and outer-polymorphism makes it possible to take many instances of the data structure and fill them with values of different types. Indeed, one can use the constructors for lists to build lists of integers, lists of arrays, or more generally, lists of α, whatever α is. No coercion of explicit type information is needed in the program. This is not true in some programming languages such as C or Java. Noticeably, the introduction of type parameters has been studied in the case of Java [BOSW98].

However, as soon as more abstraction is needed, ML is no longer sufficient. Indeed, assume the data structure is not known to the programmer: he wishes to write a function taking an unknown data structure as an argument, as well as an iterator over this data structure. Then the ML type system requires the data structure and the iterator to be monomorphic. It is no longer possible to explore the data structure using incompatible instances of the iterator.

As an example, consider a program that takes a list as an input data structure and only manipulates that list through the function `List.fold ls`. Such a program can always take the following form:

```
λ(ls)
  let fold = List.fold ls in
  a₂
```

where `ls` does not occur free in $\texttt{a}_2$. In order to offer more flexibility, the programmer may leave the underlying implementation of the data structure abstract. Then this piece of code would become a function expecting only an iterator: $\lambda\texttt{(fold)}\ \texttt{a}_2$. Consequently, it would be possible to use this function with different data structures such as lists, balanced trees, hashtables, etc. Unfortunately, such a function is not typable in ML as soon as its argument is used polymorphically. Observe that this requires only traversing the data structure using two incompatible instances of `fold`. In Chapter 12, we develop a similar example using a prototype implementation of ML$^{\mathrm{F}}$. Another related example is given Section 11.1 (page 203); it shows that the encoding of pairs in System F is not correct in ML.

First-class polymorphism (also called second-order polymorphism) makes it possible to typecheck such examples; we describe it more precisely in the next section. Then the argument `fold` can be given the polymorphic type $\forall\,(\alpha)\ (\alpha \to \beta \to \alpha) \to \alpha \to \alpha)$, where β stands for the type of elements of the data structure.

As a summary, abstraction over data structures is not possible in core-ML (although some implementations allows it at the module level). More generally, ML is usually not suitable to handle values with hidden type information:

As a first example, consider existential types, which can be used to encapsulate a value and hide part of the type information; that is, part of the structure of the value is hidden to the outside world. For instance, consider a server that receives tasks to be done asynchronously, queues them in a list, and may then execute them sequentially. The list contains tasks waiting to be executed: its type is `task list`. Now, how is the type `task` defined? In ML a task is usually only a computation waiting to be triggered, that is, a function of type `unit` → `unit`. However, it is often more convenient to describe a task as a pair of a function and an argument to be passed to the function. Then a task is a pair (f, a) whose type is an existential type: $\exists\alpha.(\alpha \to \texttt{unit}) \times \alpha$. Using this form of encapsulation, it is possible to put pairs of incompatible types into the task list. Existentials are not available in core-ML, but in some extension to the language [LO94a]. However, some explicit type information is needed both for encapsulation and opening, which will be improved with ML^{F}. Existential types can also be encoded quite elegantly using first-class polymorphism.

As a second example, objects are also a form of encapsulation. Since objects may contain any data structure, possibly hidden to the outside world, its interface (methods) must provide tools to copy, transform, or iterate over itself. Hence, polymorphic functions are usually needed in the interface. Take for example a class `dlist` implementing double-linked lists. Since the type of the list elements, say α, is only a type parameter of the class, the methods[1] `rev`, `car`, `cdr` are monomorphic. Their types are, respectively, α `dlist`, α, and α `dlist`. However, a method such as `fold` must be polymorphic: its type is $\forall\,(\beta)\ (\beta \to \alpha \to \beta) \to \beta \to \beta$. Such a type cannot be expressed in ML because first-class polymorphism is missing. In OCaml, a special extension to the language is required to provide polymorphic methods in objects [GR99].

Encoding monads, as done by Peyton Jones and Shields [JS04] also needs first-class polymorphism. Indeed, a monad type, which we write $\alpha\ m$, is usually parametric over some type variable α, and the functions of the monad interface must be polymorphic. For instance, `return` and `bind` are given the polymorphic types $\forall\,(\alpha)\ \alpha \to (\alpha\ m)$ and

---

[1] Following the tradition, `rev`, `car` and `cdr` are, respectively, the functions which reverses a list, returns the first element, and returns the list without the first element.

$\forall(\alpha,\beta)\ \alpha\ m \to (\alpha \to \beta\ m) \to \beta\ m$. Then functions taking a monad as an argument usually require second-order polymorphism.

As another example, non-uniform recursive types [Oka98] are also known to make use of polymorphic recursion. Two cases can be identified.

- The first one corresponds to finite types: take for instance the datatype definition `type` $\alpha\ T =$ `None` | `Some` $of\ \alpha \times$ (`int` T). A function defined over this datatype certainly needs to be defined for the types $\alpha\ T$ and `int` T. If polymorphic recursion is not available, it is still possible to define twice the same function: the first one is defined over the type `int` T, and the second one over $\alpha\ T$. This example shows how duplication of code may circumvent the limitations of the type system.

- The second case corresponds to infinite types: take for instance the following definition `type` $\alpha\ T =$ `None` | `Some` $of\ \alpha$ `list` T. A recursive function defined over such a datatype needs to be polymorphic. In this case, it is not possible to duplicate the code to circumvent the limitations of the type system.

All the examples above only require rank-2 polymorphism. However Peyton Jones and Shields [JS04] give an example that illustrates how a rank-3 polymorphic type may be needed by defining a recursive map function over a nested datatype: the recursive map function is decomposed into a recursive fixpoint operator and a non-recursive map function. Although this separation is quite artificial, it shows that rank-3 types are likely to appear in programs using polymorphic functions as soon as more abstraction is introduced. Indeed, adding an extra level of abstraction, such as the explicit separation mentioned here, immediately increases the rank of the types being involved.

As a conclusion of this section, we have seen that although outer polymorphism *a la* ML is usually enough to create and handle parametric data structures, some extension to the type system is needed as soon as one wishes to express more abstraction over data structures, encapsulation of values, or non-uniform datatypes. Indeed, in all these cases the lack of first-class polymorphism in ML prevents the typing of the corresponding programs.

First-class polymorphism

In a type system with first-class polymorphism, any variable may have a polymorphic type, including λ-bound variables. In the 1970's, Girard [Gir72] and Reynolds [Rey74] independently designed such a type system, called System F. Since full type inference is undecidable for System F [Wel94], some explicit type information is needed in order to typecheck expressions. As a consequence, System F is usually defined in Church style, where expressions have sufficient type information so that the typing derivation can be constructed. Three kinds of type information are found in System F expressions: type abstractions, type annotations, and type applications. *Type abstractions* are abstractions over type variables, and written $\Lambda(\alpha)\ a$. They indicate that the expression a is polymorphic in α. The type variable being abstracted (α in our example) is meant to appear in *type annotations* or *type applications*. Type annotations are types used to annotate λ-bound variables. For example, the identity is written $\Lambda(\alpha)\ \lambda(x : \alpha)\ x$ **(3)** in System F. A type abstraction is used to indicate that this function is polymorphic. The type annotation on x links the abstract type variable α to the type of x. In System F, the expression (3) has type $\forall\,\alpha \cdot \alpha \to \alpha$.[2] Intuitively, a polymorphic value represents an infinite number of types. Choosing one of these types (or a subset of these types) is called *instantiation*. In System F, instantiation is explicit via a *type application*. To pursue the example, instantiating the identity so that it can be applied to integers is done by writing `id` [`int`]. Type applications explicitly indicate how a polymorphic value is instantiated. The type argument (here `int`) is meant to replace the abstracted type variable (here α) in the body of the type abstraction, leading to the type `int` → `int`.

There is a great difference between programming in ML and programming in System F: while the former benefits from full type inference, so that no type is needed in the source program, the latter needs all the type information described above. However, System F is more expressive than ML since polymorphism is first class. In order to merge the two systems, a first approach may consist in performing a sufficient amount of type inference in System F, so that usual programs would need very little type annotations. Taking $F_{<:}$ as a basis, Cardelli has experimented such a solution [Car93]. It has lead to *local type inference* [PT98], recently improved to *colored* local type inference [OZZ01]. Both of these type systems succeed in merging subtyping with fist-class polymorphism, while providing a reasonable amount of type inference. However, both fail to type all unannotated ML programs. Besides, although local type inference seems satisfactory enough in practice, it suffers from intrinsic limitations that are believed hard to fix [HP99]. We compare this approach with ours at the end of the introduction.

A second approach consists of extending ML with higher-order types, so that all ML programs are typable without any annotations and System F can be encoded into the extended language [LO94b, Rém94, OL96, GR99]. However, the existing solutions are still limited in expressiveness and the amount of necessary type declarations keeps first-class polymorphism uneasy to use. More precisely, these solutions embed first-class polymorphism into ML monotypes using type constructors. This is called *boxed* polymorphism. Here, a polymorphic value may have either a polymorphic type, as in ML, or a boxed polymorphic type. Explicit coercions are needed to transform a polymorphic type into a boxed one. Conversely, whereas a polymorphic type is implicitly

---

[2] Note that we use two notations: a first one is $\forall\,(\alpha)\ \sigma$ for polymorphic types of ML, which we also use for types of ML^{F}; the second one is $\forall\,\alpha \cdot t$ for polymorphic types of System F.

instantiated in ML, an explicit coercion is needed as well to transform a boxed type into a polymorphic type. The amount of explicit type information differ from one system to another. For example, in Poly-ML [GR99] the coercion from boxed types to polymorphic types is a simple mark that does not depend on the types being coerced (the construct $\langle\cdot\rangle$). However, an explicit type annotation is required for coercing a polymorphic type to a boxed type. Although Poly-ML has been used to add polymorphic methods to OCaml, it is probably not suitable yet for providing user-friendly first-class polymorphism. In summary, existing solutions are in general uneasy to use due to the required amount of type information in programs. Indeed, explicit type information is needed both when a polymorphic value is boxed and when it is unboxed. Below, we compare these proposals with ML^{F}.

In this document, we follow the second approach: our goal is to type all ML programs without any type annotation, and to type all expressions of System F using type annotations if necessary. We build on the work done in Poly-ML, and we aim at the elimination of all coercions between unboxed polymorphic types and boxed polymorphic types. However, we do not expect to guess polymorphism, so that explicit type annotations are still required in some cases in order to typecheck expressions of System F. In summary, ML^{F} programs contain some type annotations, but no coercion, no type abstraction, and no type application.

Type inference

The ML language benefits from full type inference, while System F does not. Let us study more precisely what this means. In core-ML, programs consist of untyped lambda-expressions and a `let` construct. Notice that, by construction, they do not contain any type annotation. Not all the writable programs are valid ML programs: only some of them are typable, according to a set of *typing rules*. Fortunately, there exists a *type inference algorithm*, which finds whether a given program is typable or not. Moreover, this algorithm is *complete*, that is, it always terminates by telling if the given program is typable or not. In summary, the type inference problem in ML consists of finding whether a given program is typable according to the ML typing rules. It is remarkable that, by definition, the given programs do not contain any type information.

A similar approach for System F is called Curry style System F: programs are unannotated lambda-terms, that is, pure-lambda terms without any type information. Then typing rules are defined on pure lambda-terms, and a type inference problem consists of finding whether there exists a typing derivation using Curry style System F typing rules. Unfortunately, type inference in Curry style System F is undecidable in general [Wel94]. This means that it is impossible to design a complete algorithm which finds whether a given program is typable or not. This is probably why programs are usually defined in a Church style, that is, with a lot of type information. Additionally, typing rules are defined on fully annotated programs. Hence, defining type inference as we did for ML would be meaningless: since all the necessary type information is already present in programs, it is straightforward to write a type inference algorithm that simply checks if the given program is typable. In that sense, we have a complete type inference algorithm for Church style System F. It should be noticed that whereas a Church style System F program has the same structure as its typing derivation, a Curry style System F program may have many typing derivations.

Since full type inference is undecidable in Curry style System F, it is impossible to reconstruct all type abstractions, type applications and type annotations. Besides, reconstructing type annotations when only type abstractions and the location of type applications are indicated is undecidable as well [Pfe93]. In ML^{F}, we aim at the elimination of type abstractions and type applications, but not all annotated lambdas. Hence, an ML^{F} program is an ML program that might still contain some type annotations. As seen above, an annotated System F program corresponds exactly to its expected typing derivation, thus type inference is straightforward in Church style System F. On the contrary, an annotated ML^{F} program still requires some type inference: first, type annotations are not required on all λ-bound variables, thus it remains to infer the types of unannotated λ-bound variables, as it is the case in ML; second, type abstractions and type applications are not explicit, thus it remains to infer both the places where polymorphism is introduced and the places where polymorphism is instantiated. We see that the type inference problem over annotated ML^{F} programs is *a priori* not straightforward, and certainly more difficult than type inference problems over fully annotated System F programs. In this document, we consider only such type inference problems. In particular, we do not study type inference problems over ML^{F} without type annotations (that is, pure lambda-terms considered as ML^{F} programs). Nevertheless, a short discussion about the decidability of such problems is addressed while comparing ML^{F} and System F.

In summary, our ambition is to infer types for all ML programs and to infer type abstractions and type applications for all System F programs. However, our goal is not to guess type annotations, thus programs that need but lack type annotations must be immediately rejected. This is why, unlike ML, the static semantics of an ML^{F} program take type annotations into account.

Principal types

In ML, type inference relies on the following property: all the possible types of a program are instances of a single type, called its *principal type*. It is worth comparing what principal types are in the different flavors of System F. As seen above, a Church style System F program admits at most one type, which can be considered as a (trivial) principal type. Conversely, a Curry style System F program may admit many incomparable types. To see this, consider the expression $\lambda(x)\ x\ x$. Then taking $\sigma_{\mathtt{id}}$ for the type of x (and adding the necessary type application) makes the expression typable with type $\sigma_{\mathtt{id}} \to \sigma_{\mathtt{id}}$. However, taking $\forall\,\alpha \cdot \alpha$ for the type of x leads to the type $(\forall\,\alpha\cdot\alpha) \to (\forall\,\alpha\cdot\alpha)$. These two types are incomparable. Furthermore, there is no type that is both a valid type for $\lambda(x)\ x\ x$ and more general than these two types. Indeed, Curry style System F does not have principal types.

In ML$^{\mathrm{F}}$, any typable expression admits a principal type: all the possible types of a program are instances of a single type. Note that this holds for annotated programs. Like type inference, the problem of principal types is *a priori* not straightforward in annotated ML$^{\mathrm{F}}$, while it is trivial in fully annotated System F. In this document, we consider only the problem of principal types for annotated ML$^{\mathrm{F}}$ programs, and we do not consider the problem of principal types for ML$^{\mathrm{F}}$ without type annotations. As a consequence, the principal type of every typable expression depends on the explicit type annotations. It should be clear from the above discussion that this is not a limitation due to the design of ML$^{\mathrm{F}}$, but it is only a consequence of our ambitions.

To sum up, we consider the questions of type inference and principal types for annotated ML$^{\mathrm{F}}$ programs. Whereas these problems are trivial in fully annotated System F, they are not in ML$^{\mathrm{F}}$ since we aim at the inference of type abstractions and type applications. More precisely, the principal type of an expression must capture all possible type abstractions and type applications through an appropriate instance relation, as well as an appropriate syntax for types. Let us consider an example. We assume `choose` is an expression of type $\forall\,\alpha\cdot\alpha\to\alpha\to\alpha$ (it can be defined as $\lambda(x)\ \lambda(y)$ `if` *true* `then` x `else` y). Let us apply `choose` to the identity `id`: adding the omitted type abstractions and type applications, we get (at least) two typings for `choose id` in Church style System F:

$$\begin{array}{llll} & \mathtt{choose}\ [\forall\,\beta\cdot\beta\to\beta] \quad \mathtt{id} & : & (\forall\,\beta\cdot\beta\to\beta)\to(\forall\,\beta\cdot\beta\to\beta) \\ \Lambda(\beta) & \mathtt{choose}\ [\beta\to\beta] \quad (\mathtt{id}\ [\beta]) & : & \forall\,\beta\cdot(\beta\to\beta)\to(\beta\to\beta) \end{array}$$

The first typing is obtained by keeping the identity fully polymorphic. Then `choose id` is a function expecting an argument that must be as polymorphic as the identity. The second typing is obtained by first instantiating the identity to $\beta\to\beta$. Then `choose id` is a function expecting a monomorphic argument of type $\beta\to\beta$.

In ML$^{\mathrm{F}}$, we have to give a principal type to `choose id` that can capture both types by instantiation. We note that both types have the same structure $t\to t$. Actually, it is straightforward to check that all the types of `choose id` in System F have the form $\forall\,\bar\alpha\cdot t\to t$ for some type variables $\bar\alpha$ and some type t instance of $\forall\,\alpha\cdot\alpha\to\alpha$ (including $\forall\,\alpha\cdot\alpha\to\alpha$ itself). In ML$^{\mathrm{F}}$, we capture this by writing $\forall\,(\alpha\ge\sigma_{\mathtt{id}})\ \alpha\to\alpha$ for the type of `choose id`. The notation $(\alpha\ge\sigma_{\mathtt{id}})$ binds α to any instance of the identity $\sigma_{\mathtt{id}}$. Such a binding is called a *flexible* binding. By analogy, we write $\forall\,(\alpha=\sigma_{\mathtt{id}})\ \alpha\to\alpha$ for the type $\sigma_{\mathtt{id}} \rightarrow \sigma_{\mathtt{id}}$. The binding $(\alpha = \sigma_{\mathtt{id}})$ is a *rigid* binding. As suggested by the notation, the instance relation of ML^{F} allows the bound of α in $\forall\,(\alpha \geq \sigma_{\mathtt{id}})$ to be updated by any instance of $\sigma_{\mathtt{id}}$, but does not allow the bound of α in $\forall\,(\alpha = \sigma_{\mathtt{id}})$ to be instantiated. This new syntax for types, along with an appropriate instance relation, actually suffices to capture all implicit type abstractions and type applications; that is, we get principal types in ML^{F} by just following the intuition behind the notation $\forall\,(\alpha \geq \sigma)\ \sigma'$, which means "the type σ' where α is any instance of σ". In short, ML^{F} provides richer types with constraints on the bounds of variables that allow the instantiation of these variables to be delayed until we have the necessary information to instantiate them.

Compositional type inference

A principal typing [Jim96] is a typing judgment that somehow represents all possible typing judgments of the expression. In particular, it must represent all possible typing environments that allow the typing of the expression. We parenthetically observe that ML^{F} does not have principal typings, nor do System F or ML [Wel02]. As a counter example, in System F, ML, and ML^{F}, there is no way to provide a typing of $x\ x$ (under a typing environment binding x) that represents all its typings.

Let us have a closer look at $\lambda(x)\ x\ x$. This expression is not typable in ML. Besides, it is not typable in ML^{F} as such because it would require guessing the polymorphic type of x. Actually, a type annotation, *e.g.* $(x : \sigma_{\mathtt{id}})$ is necessary to typecheck the expression, and the expression $\lambda(x : \sigma_{\mathtt{id}})\ x\ x$, which we call `auto`, is typable in ML^{F}. Now, we consider a variant: instead of returning only $x\ x$, we return a pair composed of $x\ x$ and x; that is, we consider the expression $\lambda(x)\ (x\ x, x)$. It is not typable in ML^{F} either, for the reason given above, and a type annotation is needed. In ML^{F}, we allow any subexpression to be annotated, thus the expressions $\lambda(x)\ ((x : \sigma_{\mathtt{id}})\ x, x)$ $\mathbf{(\underline{1})}$ and $\lambda(x)\ (x\ x, (x : \sigma_{\mathtt{id}}))$ $\mathbf{(\underline{2})}$ are well-formed. Furthermore, annotations on λ-bound variables, which are the only kind of type annotations in System F, are syntactic sugar in ML^{F}: $\lambda(x : \sigma_{\mathtt{id}})\ (x\ x, x)$ is expanded into $\lambda(x)\ \mathtt{let}\ x = (x : \sigma_{\mathtt{id}})\ \mathtt{in}\ (x\ x, x)$ $\mathbf{(\underline{3})}$. In those three examples (1), (2), and (3), a type annotation $(x : \sigma_{\mathtt{id}})$ is given, thus there is no need to guess the polymorphic type of x. However, although (1) and (3) are typable in ML^{F}, we reject (2) in order to keep type inference compositional. Let us explain why. The typing of (2) requires the typing of $(x\ x)$ $\mathbf{(\underline{4})}$ and the typing of $(x : \sigma_{\mathtt{id}})$ $\mathbf{(\underline{5})}$. Additionally, the initial typing environment does not provide any information about the type of x. Hence, if the type inference algorithm first tries to type expression (5), it "learns" that the type of x is $\sigma_{\mathtt{id}}$. Then using this information, we expect it to succeed in typing (4). However, if the algorithm first tries to type expression (4), it can only fail because it cannot guess the polymorphic type of x. A solution to this problem would be to use backtracking in the type inference algorithm. Note, however, that the type information $(x : \sigma_{\mathtt{id}})$ may be located in an arbitrarily "deep" subexpression.

In such a situation, type inference is not only expansive but also counter-intuitive to the programmer. To avoid this in ML^{F}, we have designed the type system so that the expression (2) is not typable. In other words, the specification is kept compositional. Let us explain how the typing rules of ML^{F} allow for the typing of (1) but reject the typing of (2).

Considering the ML typing rules, the typing rule for λ-abstractions is restricted so that type inference is possible. Indeed, the typing of $\lambda(x)\ a$ only allows a monotype (that is, a non-polymorphic type) to be assigned to x. We use the same technique in ML^{F}: λ-abstractions can only introduce monotypes in the typing environment. However, the typing of (1) obviously requires x to have a polymorphic type. The trick is to assign a type variable to x, say α, and to bind α to a polymorphic type in a *prefix*. Thus, a λ-bound variable such as x is given a monotype α that *represents* a polymorphic type via the prefix. A fundamental property of the prefix is that its bindings are "abstracted", that is, hidden from the typing rules. To pursue the example, we say that the type variable α is an *abstraction* of the polymorphic type $\sigma_{\mathtt{id}}$. In order to use x as a polymorphic expression, it is first necessary to *reveal* its polymorphism, which can only be done with an explicit type annotation. Of course, the type annotation on x must correspond to the binding of α in the prefix.

All typing rules of ML^{F} are stated under a prefix. Then type variables can represent polytypes by putting the corresponding binding into the prefix. However, the information stored in the prefix cannot be used for typing the expression: an explicit type annotation is necessary to reveal the polymorphic type associated to a type variable. Note, however, that there is no need to reveal the type of an expression that is not used polymorphically. A related key feature of ML^{F} is that type variables can *always* be implicitly instantiated by polymorphic types. This can be illustrated by the application $(\lambda(x)\ x\ \mathtt{id})\ \mathtt{auto}\ (\lambda(x : \sigma_{\mathtt{id}})\ x\ x)$. This expression is typable in ML^{F} as such, that is without any type application nor any type annotation—except, of course, in the definition of `auto` itself. The type of x is polymorphic (it is the type of `auto`), but x itself is not used polymorphically. Thus, the type of x can be kept abstract so that no revelation (no type annotation) is needed. In fact, a generalization of this example is the `app` function $\lambda(f)\ \lambda(x)\ f\ x$, whose ML^{F} principal type is $\forall\,(\alpha, \beta)\ (\alpha \to \beta) \to \alpha \to \beta$. It is remarkable that whenever $a_1\ a_2$ is typable in ML^{F}, so is `app` $a_1\ a_2$, without any type annotation nor any type application. This includes, of course, cases where a_1 expects a polymorphic value as argument, such as in `app auto id`. We find such examples quite important in practice, since they model iterators (*e.g.* `app`) applied to polymorphic functions (*e.g.* `auto`) over structures holding polymorphic values (*e.g.* `id`).

To conclude, the typing rules of ML^{F} define a compositional specification. The technique is to enrich typing judgments with a prefix that binds type variables to polytypes, and to allow only explicit type annotations to reveal the polytype associated to a type variable. Conversely, it is possible to implicitly abstract a polytype by a type variable. This is formalized in ML^F by a relation under prefix called *abstraction*. Thus, ML^F enjoys three relations on types: the equivalence, abstraction, and instance relations. Actually the instance relation includes the abstraction relation, which includes the equivalence relation. The inverse of the abstraction relation is the revelation relation, which is used by type annotations to reveal the polytype abstracted by a type variable. The abstraction and revelation relations play a key role in type inference by setting a clear distinction between explicit and inferred type information, which is the essence of ML^F.

Variants of ML^F

All ML programs are typable in ML^F as such. Besides, all System F programs can be encoded into ML^F by dropping type abstractions and type applications. Thus, ML^F seems to be expressive enough. However, subject reduction does not hold in ML^F for a simple notion of reduction (non local computation of annotations would be required during reduction). Thus, we introduce an intermediate variant $\mathrm{ML}^\mathrm{F}_\star$ where type annotations are replaced by a place holder for revelation, called an *oracle*. For example, using the symbol $\star$ in place of polytypes, $\lambda(x:\star)\ x\ x$ belongs to $\mathrm{ML}^\mathrm{F}_\star$ since $\lambda(x:\sigma_{\mathtt{id}})\ x\ x$ belongs to ML^F. Since type annotations are replaced by oracles, the reduction rules can be expressed in a simple way that does not need heavy computation on type annotations. Subject reduction and progress are proved for $\mathrm{ML}^\mathrm{F}_\star$. Then type soundness follows for $\mathrm{ML}^\mathrm{F}_\star$ and, indirectly, for ML^F.

In fact, we abstract the presentation of $\mathrm{ML}^\mathrm{F}_\star$ over a collection of primitives so that ML^F can then be embedded into $\mathrm{ML}^\mathrm{F}_\star$ by treating type-annotations as an appropriate choice of primitives and disallowing annotation place holders in source terms. Thus, although our practical interest is the system ML^F, most of the technical developments are pursued in $\mathrm{ML}^\mathrm{F}_\star$.

Related works

Our work is related to all other works that aim at some form of type inference in the presence of first-class polymorphism. The closest of them is unquestionably Poly-ML [GR99], which is itself related to several other works:

Encapsulated polymorphism

Some previous proposals [Rém94, OL96] encapsulate first-class polymorphic values within datatypes. In these frameworks, a program must contain preliminary type definitions for all polymorphic types that are embedded in the program. For instance, the following program defines two flavors of `auto` and applies them to `id`:

```
type sid = Sid of ∀α. α → α
let id = Sid (λ(x) x)
let auto₁ x = match x with Sid x' → x' x'
let auto₂ x = match x with Sid x' → x' x
(auto₁ id, auto₂ id)
```

Notice the difference between `auto`$_1$ and `auto`$_2$: the former returns the unboxed identity; the latter returns the boxed identity.

We see that the type `sid` is used both as a constructor in the creation of the polymorphic value `id` (second line) and as a destructor in the lambda-abstraction (third and fourth lines). In practice, some syntactic sugar allows `auto`$_1$ to be defined as follows:

```
let auto₁ (Sid x) = x x
```

Poly-ML is an improvement over these proposals that let the coercion from monotypes to polytypes be a simple mark, independent of the type being coerced. Then polymorphic types do not need to be defined (bound to a constructor) before they are used, although it is still possible to do so. Noticeably, Poly-ML subsumes first-class polymorphism by datatype encapsulation.

The above example can now be written as follows:

```
let id = [λ(x) → x : ∀α. α → α]
let auto₁ (x:∀α. α → α) = ⟨x⟩ ⟨x⟩
let auto₂ (x:∀α. α → α) = ⟨x⟩ x
(auto₁ id, auto₂ id)
```

The explicit creation of a polymorphic value is still required at the first line. However, using a known polymorphic value (second and third lines) is easier since the coercion from the embedded value `x` to a polymorphic value is simply written $\langle$`x`$\rangle$. Notice how abstracting over a polymorphic value requires explicit type information but using it only requires the mark.

Still, Poly-ML is not yet fully satisfying since the explicit type information necessary to build a polymorphic value (at the first line) is utterly redundant: can a programmer accept to write down a type that is already inferred by the typechecker?

Odersky and Läufer's proposal [OL96] is two-folded: the first mechanism is described above; a second mechanism simultaneously allows a form of predicative quantification. This implies that the above example can also be written as follows:

```
let id = λ(x) x
let auto (x:∀α. α → α) = x x
auto id
```

This is also the way this program is written in ML^{F}. However, this form of quantification is predicative in Odersky and Läufer's proposal [OL96], whereas it is impredicative in ML^{F}. This makes a huge difference between the two systems: although both of them typecheck the expression $\lambda(x : \sigma)\ a$ for any type scheme σ, encapsulation is required to pass this function as a value in their proposal, and for example store it in a list. In ML^{F}, this function can be passed as any other value, without any form of explicit encapsulation. As another example, in Odersky and Läufer's system, if $f\ a$ typechecks, then it is not always the case that `app` $f\ a$ does. Hence, this restricted form of predicative polymorphism breaks compositionality. Noticeably, the encoding of System F given by Odersky and Läufer does not use annotated λ-abstractions at all, but only polymorphic datatypes. Actually, predicative quantification and polymorphic datatypes are two independent features that can only lead to a non-uniform programming style, as far as first-class polymorphism is concerned.

ML^{F} subsumes all of these proposals: type annotations are only required on some lambda-abstractions, and no form of explicit coercion between polymorphic types and embedded polytypes is ever needed. Additionally, ML^{F} is an unpredicative type system and the encoding of System F into ML^{F} is a straightforward local mapping of expressions which removes type abstractions and type applications. On the contrary, encodings based on polymorphic datatypes analyze the whole program, that is, the whole typing derivation; the transformation is not local and the inserted type definitions can be quite involved (cf. *lifting* of types in Odersky and Läufer's proposal [OL96]).

The encoding of System F into Poly-ML does not need these type definitions; however each polymorphic value is embedded and thus requires an explicit type annotation, like in `let id = [fun x →x :` $\forall\alpha.\ \alpha{\to}\alpha$`]`. As a consequence, the encoding of a given System F expression can be larger than the original expression. In ML^{F}, polymorphic values are implicitly created; that is, no explicit type annotation is required. Besides, the encoding of System F always produces smaller expressions.

A further extension of Odersky and Läufer's work is Peyton Jones and Shields'type inference for arbitrary-rank types [JS04], which we discuss below.

Fragments of System F

Other approaches consider some fragments of System F and try to perform type reconstruction. Rank-2 polymorphism allows for full type inference [KW94]. Interestingly, this system, which Jim calls Λ_2, types the same programs than I_2, the rank-2 intersection types system [Jim95]. As already mentioned, these approaches are not compositional because of the rank limitation. Since first-class polymorphism is precisely needed to introduce a higher level of abstraction, we think this fundamental limitation is not acceptable in our opinion: only the abstractions over ML values are possible, which is then just not enough. Besides, the inference algorithm in Λ_2 requires rewriting programs according to some non-intuitive reduction rules. Worse, this algorithm can only be applied to full programs: it is not possible to typecheck library modules independently. Noticeably, the algorithm given by Trevor Jim for I_2 has better properties, including principal typings. However, the equivalence between I_2 and Λ_2 is also shown by mean of rewriting techniques; thus, although a typing in I_2 can be inferred in a modular way, it does not give a modular typing in Λ_2.

Our treatment of annotations as type-revealing primitives also resembles retyping functions (functions whose type-erasure η-reduces to the identity) [Mit88]. However, our annotations are explicit and cover only certain forms of retyping functions, and conversely can do more. Type inference for System F modulo η-expansion is known to be undecidable as well [Wel96].

Several people have considered partial type inference for System F [JWOG89, Boe85, Pfe93, Sch98] and stated undecidability results for some particular variants. For instance Boehm [Boe85] and Pfenning [Pfe93] consider programs of System F where λ-abstractions can be unannotated, and only the locations of type applications are given, not the actual type argument. They both show that type reconstruction in this system is undecidable by encoding second-order unification. Both encodings manage to introduce an unannotated λ-abstraction whose argument is used polymorphically. This is precisely what we avoid in ML^{F}: all polymorphic λ-abstraction must be annotated, whereas type abstractions and type applications are inferred.

As another example, Schubert [Sch98] considers *sequent decision problems* in both Curry style System F and Church style System F. Sequent decision problems in Curry style System F mostly correspond to type inference in System F, as already studied by Wells [Wel94], and are known to be undecidable. An interesting problem in Church style System F consists in finding the typing environment Γ which makes a given program M typable (the term M is fully annotated). Schubert proves that this problem is undecidable in general by encoding a restricted form of second-order unification, which is shown equivalent to the problem of termination for two-counters automata. We see that although the program is fully annotated, the knowledge of the typing environment is necessary to typecheck it in a decidable way. Fortunately, this is the approach we have in ML^{F}, just like in ML. On the contrary, systems with intersection types, and more generally systems aiming at principal typings, have to infer both the type and the typing environment.

Second-order unification

Second-order unification, although known to be undecidable, has been used to explore the practical effectiveness of type inference for System F by Pfenning [Pfe88]. Despite our opposite choice, that is not to support second-order unification, there are at least two comparisons to be made. Firstly, Pfenning's work does not cover the language ML *per se*, but only the λ-calculus, since let-bindings are expanded prior to type inference. Indeed, ML is not the simply-typed λ-calculus and type inference in ML cannot, *in practice*, be reduced to type inference in the simply-typed λ-calculus after expansion of let-bindings. Secondly, one proposal seems to require annotations exactly where the other can skip them: in Pfenning's system [Pfe88], markers (but no type) annotations must replace type-abstraction and type-application nodes; conversely, this information is omitted in ML^F, but instead, explicit type information must remain for some arguments of λ-abstractions.

Our proposal is implicitly parameterized by the type instance relation and its corresponding unification algorithm. Thus, most of the technical details can be encapsulated within the instance relation. We would like to understand our notion of unification as a particular case of second-order unification. One step in this direction would be to consider a modular constraint-based presentation of second-order unification [DHKP96]. Flexible bounds might partly capture, within principal types, what constraint-based algorithms capture as partially unresolved multi-sets of unification constraints. Another example of restricted unification within second-order terms is unification under a mixed prefix [Mil92]. However, our notion of prefix and its role in abstracting polytypes is quite different. In particular, mixed prefixes mention universal and existential quantification, whereas ML^F prefixes are universally quantified. Besides, ML^F prefixes associate a bound to each variable, whereas mixed prefixes are always unconstrained.

Actually, none of the above works did consider subtyping at all. This is a significant difference with proposals based on local type inference [Car93, PT98, OZZ01] where subtyping is a prerequisite. The addition of subtyping to our framework remains to be explored.

Local type inference

The main difference between local type inference [PT98] and the approach we followed in ML^F is that the former avoids global unification and only matches types at application nodes in the program: type information can only be propagated between adjacent nodes. As a result, it provides a convenient way to handle subtyping and some form of type inference in System F. The motivation for local type inference is mainly practical. As a consequence of this approach, there is no standard translation of System F (except the identity), which could have illustrated how much annotation is required in an automatic translation. Rather, the authors give some measurement of necessary type annotations on real existing ML programs. Still, the local-inference programming style always requires "top-level" definitions to be annotated and polymorphism must be explicitly introduced. Not all ML programs are typable in these systems, even after decorating top-level definitions with type abstractions and type annotations: inner type annotations may also be needed. One difficulty arises from anonymous functions as well as so-called *hard-to-synthesize* arguments [HP99]. As an example, the application `app` f x may be untypable when f is polymorphic[3]. Principal types are ensured by finding a "best argument" each time a polymorphic type is instantiated. If no best argument can be found, the typechecker signals an error. Such errors do not exist in ML nor ML^{F}, where any typable expression has a principal type. It should be noticed that finding a "best argument", and thus inferring principal types in local type systems, is made more difficult because of the presence of subtyping.

Colored local type inference [OZZ01] is presented as an improvement over local type inference, although some terms typable in the latter are not typable in the former. It enriches type inference by allowing partial type information to be propagated.

Beyond its treatment of subtyping, local type inference also brings the idea that explicit type annotations can be propagated up and down the source tree according to fixed well-defined rules, which, at least intuitively, could be understood as a preprocessing of the source term. Such a mechanism is being used in the Glasgow Haskell Compiler, and can also be added on top of ML^{F} as well, as explained in Chapter 11.

Meanwhile, in Cambridge

Adapting ideas both from Odersky and Läufer's work and local type inference, Peyton Jones and Shields designed a type inference algorithm for arbitrary-rank types [JS04]. Their type system uses a restriction of Mitchell type containment relation [Mit88], which is contravariant in the domain of arrow types and covariant in their codomain. This is a first difference with ML^{F}, in which the instance relation is always covariant (including in the domain and codomain of arrow types). Let us look at an example taken from Peyton Jones and Shields'paper [JS04].

We assume we are given some functions g, k_1 and k_2, with the following signature:

$$\texttt{val } g \texttt{ : } ((\forall\beta.\ \beta \texttt{ list} \to \beta \texttt{ list}) \to \texttt{int}) \to \texttt{int}$$
$$\texttt{val } k_1 \texttt{ : } (\forall\alpha.\ \alpha \to \alpha) \to \texttt{int}$$
$$\texttt{val } k_2 \texttt{ : } (\texttt{int list} \to \texttt{int list}) \to \texttt{int}$$

By subsumption,

$$(\forall\alpha.\ \alpha \to \alpha) \to \texttt{int}$$

is less polymorphic than

$$(\forall\beta.\ \beta \texttt{ list} \to \beta \texttt{ list}) \to \texttt{int}$$

---

[3] The problem disappears in the uncurrified form, but uncurrifying is not always possible, or it may amount to introducing anonymous functions with an explicit type annotation.

As a consequence, $(g\ k_1)$ is ill-typed in Peyton Jones and Shields'type system, whereas $(g\ k_2)$ is well typed. In ML^F, the type associated to g is $\forall\,(\gamma = \forall\,(\beta)\ \beta\ \texttt{list} \to \beta\ \texttt{list}$ $)\ (\gamma \to \texttt{int}) \to \texttt{int}$ **(1)**. Hence, both $(g\ k_1)$ and $(g\ k_2)$ would be ill-typed, because ML^F supports no form of contravariance.

However, the type system of ML^F contains flexible bounds, which can help in this very example. More precisely, the type of g shows that it is a function expecting an argument x with type $(\forall\beta.\beta\ \texttt{list} \to \beta\ \texttt{list}) \to \texttt{int}$. Hence, x must be a function expecting an argument with type $\forall\beta.\beta\ \texttt{list} \to \beta\ \texttt{list}$. Let h be a function with this type. Then, g could be defined as follows:

```
let g (x:(∀β. β list → β list) → int) = x h
```

Actually, x is not used polymorphically, thus the following definition is also typable in ML^F:

```
let g2 x = x h
```

Besides, the principal type inferred for g_2 is $\forall\,(\gamma \geq \forall\beta.\beta\ \texttt{list} \to \beta\ \texttt{list})\ \forall\,(\delta)\ (\alpha \to \delta) \to \delta$ **(2)**. Comparing (1) and (2), we see that the rigid binding for γ is replaced by a flexible binding. Then, the type variable γ stands for $\forall\beta.\beta\ \texttt{list} \to \beta\ \texttt{list}$ or any instance of it, including $\texttt{int list} \to \texttt{int list}$. As a consequence, $(g_2\ k_2)$ typechecks in ML^F (and $(g_2\ k_1)$ is still not typable because the bound of γ cannot be instantiated to $\forall\alpha.\alpha \to \alpha$).

We see that although ML^F does not enjoy any form of contravariance, it may still typecheck some examples that, at first glance, seem to require contravariant subsumption.

Another difference between Peyton Jones and Shields'type system and ML^F is that the former is predicative. Thus, as already mentioned above, compositionality is restricted; data structures cannot hold polymorphic values; generic functions cannot be applied to polymorphic values. We think this is a considerable limitation. As a simple example, consider the expression `[auto]` (that is, `auto` in a singleton list).

Furthermore, this system borrows an idea from local type inference, namely inwards propagation of type information, which is formalized in the type-system specification. In ML^F, propagation of type annotations may be performed beforehand, as a simple purely-syntactic mechanism independently from the typing process. This leads to comparable features, such as the propagation of type information from interface files to implementation files. However, this remains a side mechanism, which only plays a minor role in ML^F.

Still, Peyton Jones and Shields's metatheory is much simpler than the one of ML^F. At the programmer's level, though, it is not clear which one is simpler to use.

Organization of the document

The expressiveness of ML^{F} is based essentially on its type system, while its typing rules are almost identical to those of ML. Such a combination of more expressive types and ML semantics leads to an expressive second-order type system. Types are defined and studied in Part I. Typing rules and dynamic semantics are defined and studied in Part II. The expressiveness of ML^{F} is considered in Part III.

Part I: Types Types are described in Chapter 1 as monotypes and polytypes. Monotypes are monomorphic types, without quantifiers. Polytypes are polymorphic types, and consist in a sequence of bindings, called a prefix, in front of a monotype. The equivalence relation ($\equiv$) and instance relation ($\sqsubseteq$) are then defined, as well as an intermediate relation, the abstraction relation ($\sqsubseteq\!\!\!\!-$). The two former relations can be viewed as extensions of the ML corresponding relations. The abstraction relation, however, has no counterpart in ML. It is a reversible relation, as equivalence, but only explicitly, that is, thanks to type annotations in the source code. It is used to distinguish between inferred information on types and explicitly given information. Chapter 2 gathers useful properties about types and relations between types. Some of these results are direct extensions of ML properties to the more general framework of ML^{F}. Some other results have no counterpart in ML[4], notably properties concerning the abstraction relation. The three relations on types induce immediately the same relations on prefixes. Prefixes extend the notion of substitution by allowing, intuitively, the substitution of type variables by polytypes. Types and substitutions are at the heart of the ML type system. Similarly, types and prefixes contain the expressive power of ML^{F}. Chapter 3 focuses on prefixes. Like properties on types, properties on prefixes usually extend known results about substitutions in ML, in a more general framework. The work done on types and prefixes is then used in Chapter 4, where the unification algorithm is stated, then shown correct and complete. As expected, the unification algorithm is an extension of the ML unification algorithm that deals with polytypes, and returns a prefix instead of a substitution.

Part II: The programming language As explained above, we define the language $\mathrm{ML}^{\mathrm{F}}_{\star}$, which consists of the expressions of ML enriched with a construction called an oracle. The difference between ML and $\mathrm{ML}^{\mathrm{F}}_{\star}$ lies mainly in their respective type system, not only in the oracle. The oracle is a simple mark on the expression that explicitly allows the reversal of the abstraction relation. Oracles do not contribute to the reduction of expressions, but they are silently propagated. The expressions and dynamic semantics of $\mathrm{ML}^{\mathrm{F}}_{\star}$ are given in Chapter 5, as well as the typing rules. The

---

[4] Actually, such results are trivial when stated in the framework of ML.

typing rules of $\mathrm{ML}^{\mathrm{F}}_{\star}$ are similar to those of ML: they are only enriched with an explicit prefix, and make use of the richer type instance relation. In ML, all bounds of the prefix are unconstrained, therefore the prefix is kept implicit. An additional typing rule is necessary for the oracle; it allows only to reverse the abstraction relation. Then exactly as in ML, a syntax-directed presentation of the typing rules is given. The goal of a type system is to ensure safety, that is, well-typed programs do not go wrong. Chapter 6 shows type safety for $\mathrm{ML}^{\mathrm{F}}_{\star}$. Type safety for ML^{F}, which is defined as $\mathrm{ML}^{\mathrm{F}}_{\star}$ without oracles, follows. As suggested by their name, oracles cannot be guessed. More precisely, oracles reverse the abstraction relation, which is safe (type safety is preserved), but breaks type inference. Indeed, Chapter 7 gives a type inference algorithm which is shown sound and complete for ML^{F}, but not for $\mathrm{ML}^{\mathrm{F}}_{\star}$, where it is probably undecidable. In summary, a richer language is introduced, namely $\mathrm{ML}^{\mathrm{F}}_{\star}$ in order to show type safety of ML^{F}. However, ML^{F} is the programming language that we wish to use, and which provides type inference. Although the syntax of ML^{F} and ML are the same, the type system of the former is much richer than the type system of the latter. However, we show in Chapter 8 that core-ML^{F} and ML are equivalent, that is, the power available in the type system of ML^{F} is not used. Then we introduce explicit type annotations as a set of primitives, which make use of the richer type system. We prove that type safety still holds. A few examples show how type-annotations primitives can be used to introduce first-class polymorphism. The type inference mechanism simply takes advantage of the information given by type annotations present in the source code. The expressiveness of ML^{F} with type annotations is addressed in the third part.

Part III: Expressiveness of ML^{F} As claimed so far, ML^{F} is a language with first-class polymorphism. We back up this statement by proposing an encoding of System F into ML^{F}. In Chapter 9, we prove that this encoding is sound, that is, every encoding of an expression typable in System F is typable in ML^{F}. Noticeably, the encoding simply removes type abstractions and type applications, thus a given program is always smaller in ML^{F} than in System F. In Chapter 10, we address the converse question: Can ML^{F} be encoded in System F? We first consider a restriction of ML^{F}, Shallow ML^{F}, where type annotations can only be types of System F. Then we show that System F can be encoded in Shallow ML^{F}. Conversely, we propose an interpretation of types of Shallow ML^{F}, and show that each typing in Shallow ML^{F} corresponds to one or many typings in System F. As a consequence, Shallow ML^{F} and System F have the same set of typable terms, up to type information. We use this equivalence between Shallow ML^{F} and System F to discuss the differences between ML^{F} and System F. Next, considering ML^{F} as a basis for a real programming language, we look at some useful extensions in Chapter 11. As a first example, references can be included in ML^{F}. However, as in ML, generalization has to be restricted, and we consider the well-known value-only polymorphism restriction. A more usable proposal by Jacques Garrigue [Gar02] is also presented and discussed. As a second example, we show how type annotations can be propagated from interface files to implementation files thanks to a straightforward mechanism. Finally, Chapter 12 addresses ML^{F} in practice. We show some examples written in ML^{F} and tested with our prototype.

Conventions

In this document, we distinguish four kinds of formal results: lemmas, corollaries, properties, and theorems. A *lemma* states a single result, which is usually used to show other results. We sometimes gather several minor results in a single statement. Each result is called a *property*, then. Properties are small results, usually used to show lemmas. A *corollary* is always found directly after a result of which it is a direct consequence. A *theorem* is a fundamental result of the thesis.

Proofs of formal results are given inline or they are moved to Appendix A. We give proofs inline by default, and move them to appendix when they are basically similar to already seen proofs, or when they are too long and of little interest. Additionally, a very small number of straightforward results are not proven in this document; the reader should be able to find the proof alone without any difficulty.

Statements with a short scope, such as intermediate results in proofs or constructions introduced locally, are sometimes labelled with an underlined number like this **(1)**. Then they can be referred at like this (1). Here is a complete example:

> Epimenides is a Cretan **(2)**. Epimenides says: “All Cretans are liars.” **(3)**. From (2) and (3), we conclude that Epimenides cannot tell the truth **(4)**, that is, there is at least one Cretan that is not a liar. Then we know from (4) that Epimenides is not this Cretan **(5)**. As a consequence of (2) and (5), there exist at least two Cretans.

We highly recommend to use Active-DVI [5] to read proofs since it provides eye candy features that really eases the reading of such materials.

Inference rules are written in small capitals, such as, for example R-Trans. Some derivable rules are also introduced. Their names always finish by a star, such as for example Eq-Mono*. All the inference rules of the document are gathered in the index of rules, page 319. Such rules are meant to build derivations, in the usual way. Unless specified otherwise, the size of a derivation is the number of rules it uses.

---

[5] `http://pauillac.inria.fr/advi/`

We bind an element to a name by using the symbol $\triangleq$. For instance, we write $\tau \triangleq \tau_1 \to \tau_2$ to mean that, from now on, τ is $\tau_1 \to \tau_2$. This is different from $\tau = \tau_1 \to \tau_2$, which means that the already-defined object τ happens to be equal to $\tau_1 \to \tau_2$.

Notation A sequence or a set is usually written $\bar{x}$. For instance, $\bar{\alpha}$ is a set of type variables. We also use the letters I or J for unordered sets of type variables. An index of all notations can be found page 322. We sometimes abbreviate "if and only if" into "iff".

Following the usual notations, applications of λ-terms associates to the left, and the scope of the binding $\lambda(x)$ extends as far to the right as possible. As for types, the arrow $\to$ associates to the right, and the bindings $\forall\,(\alpha)$ extends as far to the right as possible.

Part I

Types

Chapter 1

Types, prefixes, and relations under prefixes

As explained in the introduction, ML$^{\mathrm{F}}$ uses a new syntax for types, which embeds first-class quantification. In ML, monotypes are simple types without quantifiers and polytypes (also called type schemes) consist of a list of quantified variables $\forall\,(\bar\alpha)$ followed by a monotype, *e.g.* $\forall\,(\alpha,\beta)\ \alpha \to \beta$. In ML$^{\mathrm{F}}$ too, we make the difference between monotypes and polytypes. Monotypes are similar to monotypes of ML. Polytypes are more elaborate than in ML: the list of quantified variables provides a bound for each variable. For example, $\forall\,(\alpha_1 \geq \sigma_1, \alpha_2 = \sigma_2)$ are valid bindings in ML$^{\mathrm{F}}$. A list of variables with their bound is called a *prefix*. To pursue the example, $(\alpha_1 \geq \sigma_1, \alpha_2 = \sigma_2)$ is a valid prefix in ML$^{\mathrm{F}}$. The first quantification, binding α_1, is *flexible*: its bound can be instantiated. The second quantification, binding α_2, is *rigid*: its bound cannot be instantiated. Intuitively, rigid bounds correspond to the type of arguments which are required to be polymorphic, whereas flexible bounds correspond to the type of expressions that happen to be polymorphic, but that are not constrained to be.

Types and prefixes are introduced in Sections 1.1 and 1.2, respectively. Then Section 1.3 defines occurrences in a type, free variables, and substitutions.

The expressiveness of a type system lies in its relations between types. In ML$^{\mathrm{F}}$, we have a hierarchy made of three relations on types: equivalence ($\equiv$), abstraction ($\mathrel{-\!\!\!\sqsubset}$), and instance ($\sqsubseteq$). All these relations are defined *under prefix*, which means that we write $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ instead of simply $\sigma_1 \sqsubseteq \sigma_2$. The given prefix Q provides a bound to all free variables of σ_1 and σ_2. The equivalence, abstraction, and instance relations are described in Sections 1.5, 1.6, and 1.7, respectively. Additionally, we show in Section 1.5, that types have a unique canonical form, up to commutation of independent binders. This means that we capture the equivalence relation in a notion of canonical form.

This first chapter establishes many results (properties, lemmas or corollaries), most of which are used throughout the document. Only a very few of these are "local" in the sense they are not referenced at in other chapters.

One of the most important result is Corollary 1.5.10 (page 54), which characterizes the equivalence under prefix. Property 1.5.6.i (page 51) states the equivalence between a given type and its normal form; it is widely used throughout the thesis. A related result is 1.5.6.iii (page 51), which shows that a substitution and the normal form operator commute. Property 1.5.4.i (page 50) is also widely used; it states that equivalent types have the same structure. A standard result is Property 1.7.2.iii (page 59), which corresponds to the addition of unused hypotheses in the prefix of instance derivations. Another standard result is Property 1.7.2.i (page 59); it shows that a renaming can be applied to a whole instance derivation.

1.1 Syntax of types

The syntax of types is the following:

$$\begin{array}{ll} \tau ::= \alpha \mid g^n\ \tau_1\ ..\ \tau_n & \text{Monotypes} \\ \sigma ::= \tau \mid \bot \mid \forall\,(\alpha \geq \sigma)\ \sigma \mid \forall\,(\alpha = \sigma)\ \sigma & \text{Polytypes} \end{array}$$

We distinguish between *monotypes* and *polytypes*. By default, "types" refers to the more general form, *i.e.* to polytypes. The syntax of monotypes is parameterized by an enumerable set of type variables $\alpha \in \vartheta$ and a family of type symbols $g \in \mathcal{G}$ given with their arity $|g|$. To avoid degenerate cases, we assume that $\mathcal{G}$ contains at least a symbol of arity two (the infix arrow $\rightarrow$) and a symbol of arity zero (*e.g.* `unit`). We write g^n if g is of arity n. As shown by the definition, a monotype is either a type variable in ϑ, or a constructed type $g^n\ \tau_1\ ..\ \tau_n$. An example of constructed type is $(\rightarrow)\ \tau\ \tau'$, which we write $\tau \rightarrow \tau'$. As in ML, monotypes do not contain quantifiers.

Polytypes generalize ML type schemes. They are either a monotype τ, bottom (written $\bot$), or a binding quantified in front of a polytype, that is, $\forall\,(\alpha{\geq}\sigma)\ \sigma'$ or $\forall\,(\alpha{=}\sigma)\ \sigma'$. Inner quantifiers as in System F cannot be written directly inside monotypes. However, they can be simulated with types of the form $\forall\,(\alpha = \sigma)\ \sigma'$, which stands, intuitively, for the polytype σ' where all occurrences of α would have been replaced by the polytype σ. Noticeably, our notation contains additional meaningful sharing information. Finally, the general form $\forall\,(\alpha \geq \sigma)\ \sigma'$ intuitively stands for the collection of *all polytypes σ' where α is an instance of σ*. We say that α has a *rigid bound* in $(\alpha = \sigma)$ and a *flexible bound* in $(\alpha \geq \sigma)$. Intuitively, the polytype $\bot$ corresponds to the ML scheme $\forall\alpha.\alpha$. More precisely, it is made equivalent to $\forall\,(\alpha \geq \bot)\ \alpha$. Actually, ML type schemes can be seen as polytypes of the form $\forall\,(\alpha_1 \geq \bot)\ \ldots \forall\,(\alpha_n \geq \bot)\ \tau$ with outer quantifiers. Indeed, a particular case of flexible bound is the *unconstrained bound* $(\alpha \geq \bot)$, which we abbreviate as (α).

Notations Given a sequence $\alpha_1, \ldots \alpha_n$ of variables, we define the type $\nabla_{\alpha_1,\ldots\alpha_n}$ as an abbreviation for $\alpha_1 \to \ldots \to \alpha_n \to \texttt{unit}$. We write $\bar{\tau}$ for tuples of types, and $\bar{\alpha} \# \bar{\beta}$ means that the sets $\bar{\alpha}$ and $\bar{\beta}$ are disjoint. For convenience, we write $(\alpha \diamond \sigma)$ for either $(\alpha = \sigma)$ or $(\alpha \geq \sigma)$. The symbol $\diamond$ acts as a meta-variable and two occurrences or more of $\diamond$ in the same context mean that they stand for $=$ or all stand for $\geq$. To allow independent choices we use indices like in $\diamond_1$ and $\diamond_2$ for unrelated occurrences.

1.2 Prefixes

A polytype is a sequence of bindings, called a prefix, in front of a monotype. As remarked in the introduction, the expressiveness of types essentially comes from the expressiveness of prefixes. Indeed, the only difference between ML polytypes and ML^{F} polytypes lie in their bindings, that is, in their prefixes.

A *prefix* Q is a sequence of bindings $(\alpha_1 \diamond_1 \sigma_1) \ldots (\alpha_n \diamond_n \sigma_n)$ where variables $\alpha_1, \ldots \alpha_n$ are pairwise distinct and form the domain of Q, which we write $\mathsf{dom}(Q)$. The order of bindings in a prefix is significant: bindings are meant to be read from left to right; furthermore, we require the variables α_j not to occur in σ_i whenever $i \leq j$ (up to α-conversion of bound variables). As an example, $(\alpha \geq \bot, \beta = \alpha \to \alpha)$ is well formed, but $(\alpha = \beta \to \beta, \beta \geq \bot)$ is not. A prefix is called *unconstrained* when it is of the form $(\alpha_1 \geq \bot, \ldots \alpha_n \geq \bot)$. One can consider that in ML, the prefix is implicit and unconstrained.

Notations Since $\alpha_1, \ldots \alpha_n$ are pairwise distinct, we can unambiguously write $(\alpha \diamond \sigma) \in Q$ to mean that Q is of the form $(Q_1, \alpha \diamond \sigma, Q_2)$.

We write ∇_Q for $\nabla_{\mathsf{dom}(Q)}$, $Q \# Q'$ for $\mathsf{dom}(Q) \# \mathsf{dom}(Q')$, and $\forall\,(Q)\;\sigma$ for the type $\forall\,(\alpha_1 \diamond_1 \sigma_1) \ldots \forall\,(\alpha_n \diamond_n \sigma_n)\;\sigma$.

1.3 Occurrences

In this section, we define a projection function which maps occurrences in the domain of a given type to a type symbol such as $\bot$, a type constructor, or a type variable. We first introduce skeletons which are trees representing the projection function.

1.3.1 Skeletons

Definition 1.3.1 *Skeletons* are defined by the following grammar:

$$t ::= \alpha \mid \bot \mid g^n\; t_1 \mathrel{..} t_n$$

□

Substitutions are defined on skeletons in the obvious way. Since skeletons do not bind any variables, substitutions on skeletons are always capture-avoiding. The substitution of α by t' is written $[t'/\alpha]$. We write $t[t'/\alpha]$ for the skeleton t where α is substituted by t'. Substitutions on skeletons are written Θ.

1.3.2 Projections

An occurrence is a sequence of natural numbers. We write ϵ for the empty sequence, and k^n for the sequence $k, \dots, k$ of length n (and, in particular, k^0 is ϵ). The *projection* function maps pairs t/u composed of a skeleton t and an occurrence u to a type symbol, that is, to an element of the set $\{\bot\} \cup \vartheta \cup \mathcal{G}$. It is defined inductively by the following rules:

$$\bot/\epsilon = \bot \qquad \alpha/\epsilon = \alpha \qquad g^n\ t_1\ ..\ t_n/\epsilon = g \qquad \frac{i \in 1..n}{g^n\ t_1\ ..\ t_n/iu = t_i/u}$$

We write $t/$ for the projection function $u \mapsto t/u$. We note that the projection $t/$ is isomorphic to t.

Definition 1.3.2 The *projection* of an ML$^{\mathrm{F}}$-type σ to a skeleton, written $\mathsf{proj}(\sigma)$, is defined inductively as follows:

$$\mathsf{proj}(\tau) = \tau \qquad \mathsf{proj}(\bot) = \bot \qquad \mathsf{proj}(\forall\,(\alpha \diamond \sigma)\ \sigma') = \mathsf{proj}(\sigma')[\mathsf{proj}(\sigma)/\alpha]$$

Given a prefix Q equal to $(\alpha_1 \diamond_1 \sigma_1, \dots \alpha_n \diamond_n \sigma_n)$, we define Θ_Q as the idempotent substitution on skeletons $[\mathsf{proj}(\sigma_1)/\alpha_1] \circ \dots \circ [\mathsf{proj}(\sigma_n)/\alpha_n]$ (it is the composition of n elementary substitutions). □

We write $\sigma/$ for the function $\mathsf{proj}(\sigma)/$. We call *occurrences of a polytype* σ the domain of the function $\sigma/$, which we abbreviate as $\mathsf{dom}(\sigma)$. Notice that $\mathsf{dom}(\sigma)$ is a set of occurrences, while $\mathsf{dom}(Q)$ is a set of variables. We write $\sigma \cdot u/$ to denote the function $u' \mapsto \sigma/uu'$. Note that $\sigma \cdot \epsilon/$ is equal to the projection $\sigma/$.

Example 1.3.1 Occurrences ignore quantifiers and only retain the structure of types. For instance, the polytypes $\sigma_1 \triangleq \forall\,(\alpha \geq \bot)\ \alpha \to \alpha$ and $\sigma_2 \triangleq \forall\,(\beta = \bot)\ \forall\,(\gamma \geq \bot)\ \beta \to \gamma$ have the same occurrences, that is, $\sigma_1/ = \sigma_2/$. Equivalently, $\mathsf{proj}(\sigma_1)$ and $\mathsf{proj}(\sigma_2)$ are both equal to $\bot \to \bot$. More precisely, the domains of both σ_1 and σ_2 are equal to $\{\epsilon, 1, 2\}$ and both σ_1/ϵ and σ_2/ϵ are equal to $\to$. Additionally, $\sigma_1/1$, $\sigma_1/2$, $\sigma_2/1$ and $\sigma_2/2$ are all equal to $\bot$.

Properties 1.3.3 *The following properties hold, for any prefix Q and types σ, σ_1, σ_2:*

i) $\mathsf{proj}(\forall\,(Q)\ \sigma)$ *is* $\Theta_Q(\mathsf{proj}(\sigma))$.

ii) *If* $\sigma_1/ = \sigma_2/$, *then* $(\forall\,(Q)\ \sigma_1)/ = (\forall\,(Q)\ \sigma_2)/$.

Proof: Property i : It is shown by induction on the size of Q. If Q is $\emptyset$, then Θ_Q is the identity, and the result follows. Otherwise, Q is $(\alpha \diamond \sigma', Q')$ and $\mathsf{proj}(\forall\,(Q)\ \sigma)$ is $\mathsf{proj}(\forall\,(\alpha \diamond \sigma')\ \forall\,(Q')\ \sigma)$ by notation, that is, $\mathsf{proj}(\forall\,(Q')\ \sigma)[\mathsf{proj}(\sigma')/\alpha]$ **(<u>6</u>)** by definition. By induction hypothesis on Q', $\mathsf{proj}(\forall\,(Q')\ \sigma)$ is $\Theta_{Q'}(\mathsf{proj}(\sigma))$. Hence by (6), $\mathsf{proj}(\forall\,(Q)\ \sigma)$ is $(\Theta_{Q'}(\mathsf{proj}(\sigma')))[\mathsf{proj}(\sigma')/\alpha]$, that is, $([\mathsf{proj}(\sigma')/\alpha] \circ \Theta_{Q'})(\mathsf{proj}(\sigma'))$. We get the expected result by observing that $[\mathsf{proj}(\sigma')/\alpha] \circ \Theta_{Q'}$ is Θ_Q by definition. Property ii : We have by hypothesis $\sigma_1/ = \sigma_2/$, that is, $\mathsf{proj}(\sigma_1) = \mathsf{proj}(\sigma_2)$. Hence, $\Theta_Q(\mathsf{proj}(\sigma_1)) = \Theta_Q(\mathsf{proj}(\sigma_2))$ holds, which gives $\mathsf{proj}(\forall\,(Q)\ \sigma_1) = \mathsf{proj}(\forall\,(Q)\ \sigma_2)$ by Property i. By definition, this means that $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$ holds. ■

1.3.3 Free type variables and unbound type variables

In ML, type variables that appear in a type are either bound or free. In ML$^{\mathrm{F}}$, the set of free variables is only a subset of unbound variables. For example, consider the type σ equal to $\forall\,(\alpha = \beta \to \beta)\ \gamma \to \gamma$. Since α is bound but not used, σ is considered equivalent to $\gamma \to \gamma$. Hence, γ is free in σ, while β is only unbound. We give the two definitions for unbound variables and free variables. To ease the presentation, free variables are defined using skeletons.

Definition 1.3.4 A type variable α is *free* in σ if there exists an occurrence u such that σ/u is α. We write $\mathsf{ftv}(\sigma)$ the set of *free type variables* of σ. The set of *unbound type variables* of σ, written $\mathsf{utv}(\sigma)$, is defined inductively as follows:

$$\mathsf{utv}(\alpha) \triangleq \{\alpha\} \qquad \mathsf{utv}(g^n\ \tau_1\ ..\ \tau_n) \triangleq \bigcup_{i \in 1..n} \mathsf{utv}(\tau_i) \qquad \mathsf{utv}(\bot) \triangleq \emptyset$$

$$\mathsf{utv}(\forall\,(\alpha \diamond \sigma)\ \sigma') \triangleq \big(\mathsf{utv}(\sigma') - \{\alpha\}\big) \cup \mathsf{utv}(\sigma)$$

□

The definition of free variables is equivalent to the usual inductive definition. In particular, we have the usual properties $\mathsf{ftv}(\alpha) = \{\alpha\}$ (since $\alpha/\epsilon = \alpha$), and $\mathsf{ftv}(\tau_1 \to \tau_2) = \mathsf{ftv}(\tau_1) \cup \mathsf{ftv}(\tau_2)$ (since $(\tau_1 \to \tau_2)/u = \alpha$ holds if and only if u is of the form iv (a number i followed by a path v) such that $\tau_i/v = \alpha$). Moreover, the given definition amounts to considering that α is bound in σ', but not in σ in the polytype $\forall\,(\alpha \diamond \sigma)\ \sigma'$. That is, $\mathsf{ftv}(\forall\,(\alpha \diamond \sigma)\ \sigma')$ is $(\mathsf{ftv}(\sigma') - \{\alpha\}) \cup \mathsf{ftv}(\sigma)$ if α is free in σ', and $\mathsf{ftv}(\sigma')$ otherwise. Indeed, by definition, β is free in $\forall\,(\alpha \diamond \sigma)\ \sigma'$ if and only if there exists u such that $(\mathsf{proj}(\sigma')[\mathsf{proj}(\sigma)/\alpha])/u = \beta$. We see that β is free in $\forall\,(\alpha \diamond \sigma)\ \sigma'$ if and only if β is free in σ' and is not α, or α is free in σ' and β is free in σ. Additionally, we note that $\mathsf{utv}(\tau) = \mathsf{ftv}(\tau)$ for any monotype τ. Moreover, it is straightforward to check by structural induction that $\mathsf{ftv}(\sigma) \subseteq \mathsf{utv}(\sigma)$ holds for any σ.

By extension, the free variables of a set of types $\{\sigma_1, \ldots \sigma_n\}$ is defined as the union $\bigcup_{i=1..n} \mathsf{ftv}(\sigma_i)$. Similarly, we write $\mathsf{utv}(\sigma_1, \ldots, \sigma_n)$ for the union $\bigcup_{i=1..n} \mathsf{utv}(\sigma_i)$. A polytype is *closed* if it has no unbound type variable. When $\mathsf{utv}(\sigma) \subseteq \mathsf{dom}(Q)$, we also say that σ is closed under prefix Q. We define $\mathsf{utv}(Q)$ as $\mathsf{utv}(\forall\,(Q)\ \bot)$. When $\mathsf{utv}(Q)$ is empty, that is, $\forall\,(Q)\ \bot$ is closed, we say that Q is a closed prefix.

α-conversion Polytypes are considered equal modulo α-conversion where $\forall\,(\alpha \diamond \sigma)\ \sigma'$ binds α in σ', but not in σ. Let Q be $(\alpha_1 \diamond_1 \sigma_1, \ldots \alpha_n \diamond_n \sigma_n)$. Note that the α_i's can be renamed in the type $\forall\,(Q)\ \sigma$, but not in the stand-alone prefix Q. For instance, $\forall\,(\alpha \geq \bot)\ \alpha \to \alpha$ is considered equal to $\forall\,(\beta \geq \bot)\ \beta \to \beta$, but $(\alpha \geq \bot)$ and $(\beta \geq \bot)$ are distinct prefixes. We easily check that free variables and unbound variables are stable under α-conversion.

1.3.4 Renamings and substitutions

Given a function f from type variables to monotypes (typically a renaming or a substitution, as defined next), we define the domain of f, written $\mathsf{dom}(f)$, as the set of type variables α such that $f(\alpha) \neq \alpha$. The codomain of f, written $\mathsf{codom}(f)$, is the set $\mathsf{ftv}(f(\mathsf{dom}(f)))$, that is, $\bigcup_{\alpha \in \mathsf{dom}(f)} \mathsf{ftv}(f(\alpha))$. Such functions are extended to types and type schemes in the obvious way, by avoiding capture of bound variables. An idempotent function f is such that $f \circ f = f$, this amounts to having $\mathsf{dom}(f) \# \mathsf{codom}(f)$. The composition $f_1 \circ f_2$ is well-formed if we have $\mathsf{codom}(f_1) \# \mathsf{dom}(f_2)$. A function is said *invariant* on $\bar\alpha$ whenever $\mathsf{dom}(f) \# \bar\alpha$.

A *renaming* ϕ is an idempotent function mapping type variables to type variables that is injective on the finite set $\mathsf{dom}(\phi)$. Renamings are bijective from their domain to their codomain, but are never bijective on ϑ (except the identity). A renaming ϕ is said *disjoint* from $\bar\alpha$ whenever $\bar\alpha \# \mathsf{dom}(\phi) \cup \mathsf{codom}(\phi)$ holds. If ϕ is a renaming, we write $\phi^\neg$ for the inverse renaming, whose domain is $\mathsf{codom}(\phi)$ and codomain is $\mathsf{dom}(\phi)$. More precisely, for every β in $\mathsf{codom}(\phi)$, we define $\phi^\neg(\beta)$ as α such that $\phi(\alpha) = \beta$. We say that ϕ is a renaming of a set $\bar\alpha$ if ϕ is a renaming and if $\mathsf{dom}(\phi) \subseteq \bar\alpha$ and $\mathsf{codom}(\phi) \# \bar\alpha$ hold. For example, the swapping $\alpha \mapsto \beta, \beta \mapsto \alpha$ is bijective on ϑ but is not a renaming. The mapping $\alpha \mapsto \beta$ is a renaming, though, and its inverse $\beta \mapsto \alpha$ is a renaming too. Note that $\phi^\neg \circ \phi$ is not the identity, but is $\phi^\neg$ (which is invariant on $\mathsf{dom}(\phi)$, though). Conversely, $\phi \circ \phi^\neg$ is ϕ.

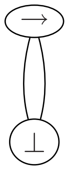
A *substitution* θ is an idempotent function mapping type variables to monotypes such that $\mathsf{dom}(\theta)$ is finite. Any renaming is also a substitution. We write *id* for the identity on type variables, considered as a substitution.

Notation The capture-avoiding substitution of α by τ in σ is written $\sigma[\tau/\alpha]$. Given a prefix Q equal to $(\alpha_i \diamond_i \sigma_i)^{i \in I}$, and a substitution θ, we define $\theta(Q)$ as $(\theta(\alpha_i) \diamond_i \theta(\sigma_i))$

provided that θ restricted to $\mathsf{dom}(Q)$ is a renaming of $\mathsf{dom}(Q)$. In particular, if ϕ is a renaming of $\mathsf{dom}(Q)$, then $\phi(Q)$ is a well-formed prefix. Note also that $\forall\,(\phi(Q))\ \phi(\tau)$ is alpha-convertible to $\forall\,(Q)\ \tau$. Additionally, if θ is a substitution such that $\mathsf{dom}(Q)\ \#\ \mathsf{dom}(\theta) \cup \mathsf{codom}(\theta)$ holds, then $\theta(Q)$ is the prefix $(\alpha_i \diamond_i \theta(\sigma_i))^{i \in I}$.

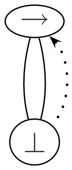
Example 1.3.2 As already mentioned, types only allows quantifiers to be outermost, or in the bound of other bindings. Therefore, the type $\forall\,\alpha \cdot (\forall\,\beta \cdot (\tau[\beta] \to \alpha)) \to \alpha$ of System F cannot be written directly. (Here, $\tau[\beta]$ means a type τ in which the variable β occurs.) However, it can be represented in ML^{F} by the type $\forall\,(\alpha)\ \forall\,(\beta' = \forall\,(\beta)\ \tau[\beta] \to \alpha)\ \beta' \to \alpha$. In fact, all types of System F can easily be represented as polytypes by recursively binding all occurrences of inner polymorphic types to fresh variables beforehand (an encoding from System F into ML^{F} is given in Section 9.2). In this translation, auxiliary variables are used in a linear way. For instance, $(\forall\,\alpha \cdot \tau) \to (\forall\,\alpha \cdot \tau)$ is translated into $\forall\,(\alpha_1 = \forall\,(\alpha)\ \tau)\ \forall\,(\alpha_2 = \forall\,(\alpha)\ \tau)\ \alpha_1 \to \alpha_2$. Intuitively, $\forall\,(\alpha' = \forall\,(\alpha)\ \tau)\ \alpha' \to \alpha'$ could also represent the same System F type, more concisely. However, this type is not equivalent to the previous one, but only an abstraction of it, as explained further.

Graphs For illustration purposes, we represent types as graphs throughout this document. More precisely, a type can be represented by two DAGs (Directed Acyclic Graphs) sharing the same set of nodes. The nodes are labelled with either a type symbol g, $\bot$, or a type variable. The first DAG corresponds to the symbolic structure of a type. The second DAG corresponds to its binding structure. For example, we consider the type of the identity, $\sigma_{\mathtt{id}}$, equal to $\forall\,(\alpha \geq \bot)\ \alpha \to \alpha$. Its skeleton is $\bot \to \bot$. Such a representation ignores that the left-hand side and the right-side of the arrow correspond to the same variable α. In graphs, we make this information explicit by sharing the two nodes, as shown below:

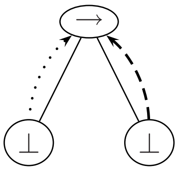


In this graph, we show explicitly that the left-hand child and the right-hand child of the arrow are both the type variable α. The bottom node represents α, and is therefore labelled by the bound of α, that is $\bot$.

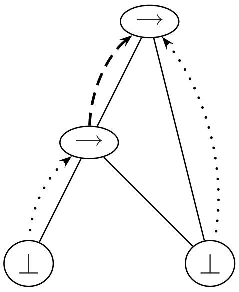
It remains to represent the binding structure of the type. We use dotted arrows and dashed arrows to represent, respectively, flexible bindings and rigid bindings. To pursue the above example, the node labelled by $\bot$, which corresponds to the type variable α, is bound at top-level. This binding is flexible, thus it is represented by a dotted arrow:



For the sake of comparison, the type $\forall\,(\alpha \geq \bot, \beta = \bot)\ \alpha \to \beta$ is represented by the following graph, where α is represented by the left-hand node, and β by the right-hand node.



As another example, the representation of $\forall\,(\alpha \geq \bot)\ \forall\,(\beta = \forall\,(\gamma \geq \bot)\ \gamma \to \alpha)\ \beta \to \alpha$ is given below. The middle node represents β, it is rigid and bound at top-level, hence the dashed arrow. The leftmost node is γ, it is bound at the level of β (that is, γ is introduced syntactically in the bound of β) and it is flexible, hence the dotted arrow from this node to the middle node representing β. The rightmost node is α, it is flexible and bound at top-level.



Although graphs are only a convenient representation of types, they may be used as a basis for an implementation.

Types are meant to be instantiated implicitly as in ML along an instance relation $\sqsubseteq$. As explained in the introduction, we also consider an abstraction relation $\sqsubseteq\!\!\!\!-$ and its inverse relation $\exists$ that is used by annotations to reveal a polymorphic type. In Section 1.5, we first define an equivalence relation between types, which is the kernel of both $\sqsubseteq\!\!\!\!-$ and $\sqsubseteq$. In Section 1.6, we define the relation $\sqsubseteq\!\!\!\!-$. The instance relation $\sqsubseteq$, which contains $\sqsubseteq\!\!\!\!-$, is defined in Section 1.7. All these relations have common properties captured by the notion of *relations under prefix*, as defined in the following section.

1.4 Relations under prefix

The equivalence, abstraction, and instance relation of ML$^{\mathrm{F}}$, as well as auxiliary relations introduced later in the thesis, are all relations under prefix. The prefix is meant to bind all unbound variables of considered types. In the following definition, $\Diamond$ is meant to be replaced by either $\equiv$, $\mathrel{\sqsubset\!\!\!\!-}$, or $\sqsubseteq$.

Definition 1.4.1 A *relation under prefix* $\Diamond$ is a relation on triples composed of a well-formed prefix Q and two types σ_1 and σ_2 such that $\mathsf{utv}(\sigma_1, \sigma_2) \subseteq \mathsf{dom}(Q)$. It is written $(Q)\ \sigma_1 \Diamond \sigma_2$. □

By notation $\sigma_1 \Diamond \sigma_2$ means $(Q)\ \sigma_1 \Diamond \sigma_2$ for some unconstrained prefix Q such that $\mathsf{utv}(\sigma_1, \sigma_2) \subseteq \mathsf{dom}(Q)$.

The relations under prefix we consider in this document are all meant to be used transitively. While some of these relations are transitive by definition (such as the equivalence, abstraction, and instance relations), some other relations are first defined non-transitively, and then extended with transitivity. Hence the following definition:

Definition 1.4.2 A relation under prefix $\Diamond$ is said *transitive* if and only if it satisfies the following rule:

$$\textsc{R-Trans}\qquad \frac{(Q)\ \sigma_1 \Diamond \sigma_2 \qquad (Q)\ \sigma_2 \Diamond \sigma_3}{(Q)\ \sigma_1 \Diamond \sigma_3}$$

Given a relation $\Diamond$, we write $\Diamond^*$ for its transitive closure, that is, $\Diamond^*$ is the smallest relation containing $\Diamond$ and satisfying Rule R-Trans. □

Relations under prefix are also meant to operate under the leading prefix of a type. For example, if $\sigma_1 \Diamond \sigma_2$ holds, we expect $\forall\,(Q)\ \sigma_1 \Diamond \forall\,(Q)\ \sigma_2$ to hold too. This can be captured by a single rule:

$$\textsc{R-Context-R}\qquad \frac{(Q, \alpha \diamond \sigma)\ \sigma_1 \Diamond \sigma_2 \qquad \alpha \notin \mathsf{dom}(Q)}{(Q)\ \forall\,(\alpha \diamond \sigma)\ \sigma_1 \Diamond \forall\,(\alpha \diamond \sigma)\ \sigma_2}$$

Some rules can also operate on the bounds of the leading prefix. For example, if $\sigma_1 \Diamond \sigma_2$ holds, we may have $\forall\,(\alpha \geq \sigma_1)\ \sigma \Diamond \forall\,(\alpha \geq \sigma_2)\ \sigma$. A similar example could be given with a rigid bound instead of a flexible bound. This is captured by the two following rules

$$\textsc{R-Context-Flexible}\qquad \frac{(Q)\ \sigma_1 \Diamond \sigma_2}{(Q)\ \forall\,(\alpha \geq \sigma_1)\ \sigma \Diamond \forall\,(\alpha \geq \sigma_2)\ \sigma}$$

$$\textsc{R-Context-Rigid}\qquad \frac{(Q)\ \sigma_1 \Diamond \sigma_2}{(Q)\ \forall\,(\alpha = \sigma_1)\ \sigma \Diamond \forall\,(\alpha = \sigma_2)\ \sigma}$$

Figure 1.1: Type equivalence under prefix

Eq-Refl

$$(Q)\ \sigma \equiv \sigma$$

Eq-Free

$$\frac{\alpha \notin \mathsf{ftv}(\sigma_1)}{(Q)\ \forall\,(\alpha \diamond \sigma)\ \sigma_1 \equiv \sigma_1}$$

Eq-Comm

$$\frac{\alpha_1 \notin \mathsf{ftv}(\sigma_2) \qquad \alpha_2 \notin \mathsf{ftv}(\sigma_1) \qquad \alpha_1 \neq \alpha_2}{(Q)\ \forall\,(\alpha_1 \diamond_1 \sigma_1)\ \forall\,(\alpha_2 \diamond_2 \sigma_2)\ \sigma \equiv \forall\,(\alpha_2 \diamond_2 \sigma_2)\ \forall\,(\alpha_1 \diamond_1 \sigma_1)\ \sigma}$$

Eq-Var

$$(Q)\ \forall\,(\alpha \diamond \sigma)\ \alpha \equiv \sigma$$

Eq-Mono

$$\frac{(\alpha \diamond \sigma_0) \in Q \qquad (Q)\ \sigma_0 \equiv \tau_0}{(Q)\ \tau \equiv \tau[\tau_0/\alpha]}$$

Definition 1.4.3 A relation under prefix $\Diamond$ is said flexible-compliant iff it satisfies rules R-Context-Flexible and R-Context-R. It is said rigid-compliant iff it satisfies rules R-Context-Rigid and R-Context-R. It is said congruent iff it satisfies these three rules. □

When a relation is congruent, it satisfies the following rule, which stands for both R-Context-Flexible and R-Context-Rigid, thanks to the meta-variable $\diamond$.

R-Context-L

$$\frac{(Q)\ \sigma_1 \Diamond \sigma_2}{(Q)\ \forall\,(\alpha \diamond \sigma_1)\ \sigma \Diamond \forall\,(\alpha \diamond \sigma_2)\ \sigma}$$

The three following sections define the equivalence, abstraction, and instance relations, respectively. As expected, the equivalence is a congruent relation. The abstraction relation happens to be a rigid-compliant relation, and the instance relation is a flexible-compliant relation.

1.5 Type equivalence

The order of quantifiers and some other syntactical notations are not always meaningful. Such syntactic artifacts are captured by a notion of type equivalence. As expected, the equivalence relation is symmetric, that is, if $(Q)\ \sigma_1 \equiv \sigma_2$ holds, then $(Q)\ \sigma_2 \equiv \sigma_1$ holds too. Additionally, it is usually not possible to read information from the prefix, except for monotypes. Thus, for instance $(\alpha = \tau)\ \alpha \equiv \tau$ holds (read: under the prefix $(\alpha = \tau)$, the types α and τ are equivalent). This is not true when τ is replaced by a type scheme σ.

Definition 1.5.1 (Equivalence) The *equivalence under prefix* $\equiv$ is the smallest symmetric, transitive, and congruent relation under prefix that satisfies the rules of Figure 1.1. □

Rule Eq-Comm allows the reordering of independent binders; Rule Eq-Free eliminates unused bound variables. Rule Eq-Mono allows the *reading* of the bound of a variable from the prefix when it is (equivalent to) a monotype. An example of use of Eq-Mono is $(Q, \alpha = \tau_0, Q')\ \alpha \to \alpha \equiv \tau_0 \to \tau_0$. Rule Eq-Mono makes no difference between $\geq$ and $=$ whenever the bound is equivalent to a monotype. The restriction of Rule Eq-Mono to the case where σ_0 is equivalent to a monotype is required for the well-formedness of the conclusion. Moreover, it also disallows $(Q, \alpha = \sigma_0, Q')\ \alpha \equiv \sigma_0$: variables with polymorphic bounds must be treated abstractly and cannot be silently expanded. In particular, we do not wish $(Q)\ \forall\,(\alpha{=}\sigma_0, \alpha'{=}\sigma_0)\ \alpha \to \alpha' \equiv \forall\,(\alpha{=}\sigma_0)\ \alpha \to \alpha$ to hold.

Rule Eq-Var expands into both $\forall\,(\alpha = \sigma)\ \alpha \equiv \sigma$ and $\forall\,(\alpha \geq \sigma)\ \alpha \equiv \sigma$. The former captures the intuition that $\forall\,(\alpha{=}\sigma)\ \sigma'$ stands for $\sigma'[\sigma/\alpha]$, which however, is not always well-formed. The latter may be surprising: it states that σ is an instance of $\forall\,(\alpha{\geq}\sigma)\ \alpha$, and conversely. On the one hand, the interpretation of $\forall\,(\alpha \geq \sigma)\ \alpha$ as the set of all α where α is an instance of σ implies (intuitively) that σ itself is an instance of $\forall\,(\alpha{\geq}\sigma)\ \alpha$. On the other hand, however, it is not immediate to consider $\forall\,(\alpha \geq \sigma)\ \alpha$, that is, the set of all instance of σ, as an instance of σ. Actually, we could remove this second part of the equivalence without changing the set of typable terms. However, it is harmless and allows for a more uniform presentation.

Reasoning under prefixes makes it possible to break a polytype $\forall\,(Q)\ \sigma$ and "look inside under prefix Q". For instance, it follows from iterations of Rule R-Context-R that $(Q)\ \sigma \equiv \sigma'$ suffices to show $(\emptyset)\ \forall\,(Q)\ \sigma \equiv \forall\,(Q)\ \sigma'$.

Definition 1.5.1 mentions that $\equiv$ is a symmetric relation. Therefore, in the proofs, we implicitly assume that each rule defining equivalence is symmetric. Actually, it suffices to consider only the symmetric variants of rules Eq-Var, Eq-Mono, and Eq-Free. Indeed, the other rules are already symmetric by construction.

It is not necessary to bind explicitly a monotype, as in $\forall\,(\alpha = \tau)\ \alpha \to \alpha$. Indeed, this type is equivalent to $\tau \to \tau$. More generally, the following rule is derivable with rules R-Context-R, R-Context-L, Eq-Mono, R-Trans, and Eq-Free:

Eq-Mono*

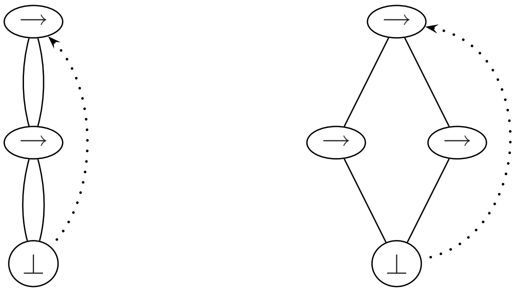
$$\frac{(Q)\ \sigma' \equiv \tau}{(Q)\ \forall\,(\alpha \diamond \sigma')\ \sigma \equiv \sigma[\tau/\alpha]}$$

Conversely, replacing Eq-Mono by Eq-Mono* would restrict the set of judgments. Indeed, whereas Eq-Mono is derivable by Eq-Mono* under an empty prefix, it is not the case under an arbitrary prefix. For a counter-example, $(\beta, \alpha = \beta \to \beta)\ \alpha \equiv \beta \to \beta$ holds by Eq-Mono, but it is not derivable if only Eq-Mono* is allowed and not Eq-Mono.

In graphs, monotypes are nodes not binding any other node. Rules Eq-Mono and Eq-Mono* state that such nodes can be equivalently duplicated. For example, the following equivalence holds by Eq-Mono*:

$$\forall\,(\alpha \geq \bot)\ \forall\,(\beta = \alpha \to \alpha)\ \beta \to \beta \equiv \forall\,(\alpha \geq \bot)\ (\alpha \to \alpha) \to (\alpha \to \alpha)$$

This means that the two following graphs are considered equivalent:



All other equivalence rules, namely Eq-Refl, Eq-Free, Eq-Var, and Eq-Comm are already captured by graphs.

The following rule is derivable with rules Eq-Free and Eq-Var:

Eq-Var*

$$\frac{(\alpha \diamond \sigma) \in Q}{(Q_0)\ \forall\,(Q)\ \alpha \equiv \forall\,(Q)\ \sigma}$$

1.5.1 Rearrangements

Normal forms will be unique, up to commutation of independent binders, that is, up to Rule Eq-Comm. We define an equivalence relation that captures only such a reordering of binders.

Definition 1.5.2 The relation $\approx$ is the smallest equivalence relation satisfying the following rules:

$$\frac{\sigma_1 \approx \sigma_2}{\forall\,(\alpha \diamond \sigma)\ \sigma_1 \approx \forall\,(\alpha \diamond \sigma)\ \sigma_2} \qquad \frac{\sigma_1 \approx \sigma_2}{\forall\,(\alpha \diamond \sigma_1)\ \sigma \approx \forall\,(\alpha \diamond \sigma_2)\ \sigma}$$

$$\frac{\alpha_1 \notin \mathsf{ftv}(\sigma_2) \qquad \alpha_2 \notin \mathsf{ftv}(\sigma_1) \qquad \alpha_1 \neq \alpha_2}{\forall\,(\alpha_1 \diamond_1 \sigma_1)\ \forall\,(\alpha_2 \diamond_2 \sigma_2)\ \sigma \approx \forall\,(\alpha_2 \diamond_2 \sigma_2)\ \forall\,(\alpha_1 \diamond_1 \sigma_1)\ \sigma}$$

When $\sigma_1 \approx \sigma_2$ holds, we say that σ_1 is a rearrangement of σ_2. When $\forall\,(Q_1)\,\bot \approx \forall\,(Q_2)\,\bot$ holds, we say that Q_1 is a rearrangement of Q_2. □

Note that the relation $\forall\,(Q_1)\,\nabla_{Q_1} \approx \forall\,(Q_2)\,\nabla_{Q_1}$ mentions ∇_{Q_1} (that is, a type mentioning all type variables of $\mathsf{dom}(Q_1)$) on both sides. Thus, it prevents implicit α-conversion of the domain of Q_1 between $\forall\,(Q_1)\,\nabla_{Q_1}$ and $\forall\,(Q_2)\,\nabla_{Q_1}$.

Properties 1.5.3

i) *If* $\sigma \approx \tau$, *then* $\sigma = \tau$.

ii) *If* $\sigma_1 \approx \sigma_2$ *holds and* θ *is a substitution, then* $\theta(\sigma_1) \approx \theta(\sigma_2)$ *holds.*

iii) *If* $\sigma_1 \approx \sigma_2$, *then* $(Q)\ \sigma_1 \equiv \sigma_2$ *holds for any well-formed prefix* Q *such that we have* $\mathsf{utv}(\sigma_1, \sigma_2) \subseteq \mathsf{dom}(Q)$

iv) *If* $Q' \approx Q$ *and* $(Q)\ \sigma_1 \equiv \sigma_2$ *hold, then* $(Q')\ \sigma_1 \equiv \sigma_2$.

v) *If* QQ' *is well-formed and* $(Q)\ \sigma_1 \equiv \sigma_2$ *holds, then* $(QQ')\ \sigma_1 \equiv \sigma_2$ *holds.*

Proof: Property i: It only states that any rearrangement of a monotype τ is τ itself.

Property ii: It is shown by induction on the derivation of $\sigma_1 \approx \sigma_2$.

Property iii: By definition $\approx$ is a binary relation. Let $\dot{\approx}$ be the relation on triples defined as $(Q)\ \sigma_1 \dot{\approx} \sigma_2$ if and only if $\sigma_1 \approx \sigma_2$ holds and Q is well-formed. Then $\dot{\approx}$ is the smallest relation satisfying rules Eq-Refl, R-Trans, R-Context-R, R-Context-L, and Eq-Comm (where $\equiv$ is replaced by $\dot{\approx}$). Consequently, $\dot{\approx}$ is included in $\equiv$. Therefore, if $\sigma_1 \approx \sigma_2$ holds, then $(Q)\ \sigma_1 \equiv \sigma_2$ holds for any well-formed Q binding the variables of $\mathsf{utv}(\sigma_1, \sigma_2)$.

Property iv : It is shown by induction on the derivation of $(Q)\ \sigma_1 \equiv \sigma_2$. Cases Eq-Refl, Eq-Free, Eq-Comm, and Eq-Var are immediate. Cases R-Trans and R-Context-L are by induction hypothesis. Case R-Context-R is by induction hypothesis, observing that $(Q, \alpha \diamond \sigma_0)$ is a rearrangement of $(Q', \alpha \diamond \sigma_0)$. As for Eq-Mono, σ_1 is τ, σ_2 is $\tau[\tau_0/\alpha]$, and the premises are $(\alpha \diamond \sigma_0) \in Q$ and $(Q)\ \sigma_0 \equiv \tau_0$. By induction hypothesis, we have $(Q')\ \sigma_0 \equiv \tau_0$ **(1)**. Since Q' is a rearrangement of Q, we have $(\alpha \diamond \sigma'_0) \in Q'$, where σ'_0 is a rearrangement of σ_0. Hence, $(Q')\ \sigma_0 \equiv \sigma'_0$ **(2)** holds by property iii. Consequently, $(Q')\ \sigma'_0 \equiv \tau_0$ holds by R-Trans, (1), and (2). Then $(Q')\ \sigma_1 \equiv \sigma_2$ holds by Rule Eq-Mono.

Property v : It is a subcase of the following, more general property:

> *If* $Q_1Q_2Q_3$ *is well-formed, and if* $(Q_1Q_3)\ \sigma_1 \equiv \sigma_2$ *holds, then* $(Q_1Q_2Q_3)\ \sigma_1 \equiv \sigma_2$ *holds.*

It is shown by induction on the derivation of $(Q_1Q_3)\ \sigma_1 \equiv \sigma_2$. ■

1.5.2 Occurrences and equivalence

Occurrences—and therefore free type variables—are stable under type equivalence:

Properties 1.5.4

i) *If* $(QQ')\ \sigma_1 \equiv \sigma_2$ *holds and* Q *is unconstrained, then* $\forall\,(Q')\ \sigma_1/ = \forall\,(Q')\ \sigma_2/$.

ii) *We have* $\tau_1 \equiv \tau_2$ *iff* τ_1 *is* τ_2.

iii) *If* $\sigma_1 \equiv \sigma_2$ *holds, then* $\mathsf{ftv}(\sigma_1) = \mathsf{ftv}(\sigma_2)$.

Proof: Property i: By definition of occurrences, it suffices to show that whenever (QQ') $\sigma_1 \equiv \sigma_2$ $(\underline{\mathbf{1}})$ holds, then $\mathsf{proj}(\forall\,(Q')\ \sigma_1)$ and $\mathsf{proj}(\forall\,(Q')\ \sigma_2)$ are equal. The proof is by induction on the derivation of (1). Case R-Trans is by induction hypothesis. In Cases Eq-Refl, Eq-Free, Eq-Comm, and Eq-Var, we have $\mathsf{proj}(\sigma_1) = \mathsf{proj}(\sigma_2)$, thus we get the expected result by Property 1.3.3.ii (page 40). Remaining cases are:

∘ Case R-Context-L: We have $\sigma_1 = \forall\,(\alpha \diamond \sigma_1')\ \sigma$ $(\underline{\mathbf{2}})$, $\sigma_2 = \forall\,(\alpha \diamond \sigma_2')\ \sigma$ $(\underline{\mathbf{3}})$, and the premise is $(QQ')\ \sigma_1' \equiv \sigma_2'$. By α-conversion, we can freely assume that α is not in $\mathsf{dom}(QQ')$. We have $\mathsf{proj}(\forall\,(Q')\ \sigma_1') = \mathsf{proj}(\forall\,(Q')\ \sigma_2')$ by induction hypothesis, that is, $\Theta_{Q'}(\mathsf{proj}(\sigma_1')) = \Theta_{Q'}(\mathsf{proj}(\sigma_2'))$ $(\underline{\mathbf{4}})$. Hence, we have the following, where i can be 1 or 2:

$$\begin{array}{rcll}
\mathsf{proj}(\forall\,(Q')\ \sigma_i) & = & \mathsf{proj}(\forall\,(Q')\ \forall\,(\alpha \diamond \sigma_i')\ \sigma) & \text{by (2) or (3)} \\
& = & \Theta_{Q'}(\mathsf{proj}(\forall\,(\alpha \diamond \sigma_i')\ \sigma)) & \text{by Property 1.3.3.i (page 40)} \\
& = & \Theta_{Q'}(\mathsf{proj}(\sigma)[\mathsf{proj}(\sigma_i')/\alpha]) & \text{by Definition 1.3.2} \\
& = & \Theta_{Q'}(\mathsf{proj}(\sigma))[\Theta_{Q'}(\mathsf{proj}(\sigma_i'))/\alpha] & (\underline{\mathbf{5}})
\end{array}$$

Additionally, by (4), we get

$$\Theta_{Q'}(\mathsf{proj}(\sigma))[\Theta_{Q'}(\mathsf{proj}(\sigma_1'))/\alpha] = \Theta_{Q'}(\mathsf{proj}(\sigma))[\Theta_{Q'}(\mathsf{proj}(\sigma_2'))/\alpha] \quad (\underline{\mathbf{6}})$$

Hence, $\mathsf{proj}(\forall\,(Q')\ \sigma_1) = \mathsf{proj}(\forall\,(Q')\ \sigma_2)$ holds by (5) and (6).

∘ Case R-Context-R: We have $\sigma_1 = \forall\,(\alpha \diamond \sigma)\ \sigma_1'$, $\sigma_2 = \forall\,(\alpha \diamond \sigma)\ \sigma_2'$, and the premise is $(QQ', \alpha \diamond \sigma)\ \sigma_1' \equiv \sigma_2'$. By induction hypothesis, we get $\mathsf{proj}(\forall\,(Q', \alpha \diamond \sigma)\ \sigma_1') = \mathsf{proj}(\forall\,(Q', \alpha \diamond \sigma)\ \sigma_2')$, which is the expected result.

∘ Case Eq-Mono: We have $\sigma_1 = \tau$ and $\sigma_2 = \tau[\tau_0/\alpha]$. The premises are $(QQ')\ \sigma_0 \equiv \tau_0$ $(\underline{\mathbf{7}})$ and $(\alpha \diamond \sigma_0) \in QQ'$. Necessarily, $(\alpha \diamond \sigma_0)$ is in Q' $(\underline{\mathbf{8}})$ because Q is unconstrained by hypothesis. By induction hypothesis and (7), we get $\mathsf{proj}(\forall\,(Q')\ \sigma_0) = \mathsf{proj}(\forall\,(Q')\ \tau_0)$, which gives $\Theta_{Q'}(\mathsf{proj}(\sigma_0)) = \Theta_{Q'}(\mathsf{proj}(\tau_0))$ $(\underline{\mathbf{9}})$ by Property 1.3.3.i (page 40). We have

$$\begin{array}{rcll}
\mathsf{proj}(\forall\,(Q')\ \tau) & = & \Theta_{Q'}(\tau) & \text{by Property 1.3.3.i (page 40)} \\
& = & \Theta_{Q'}(\tau[\Theta_{Q'}(\mathsf{proj}(\sigma_0))/\alpha]) & \text{by (8) and Definition 1.3.2} \\
& = & \Theta_{Q'}(\tau[\Theta_{Q'}(\mathsf{proj}(\tau_0))/\alpha]) & \text{by (9)} \\
& = & \Theta_{Q'}(\tau[\tau_0/\alpha]) & \text{by Definition 1.3.2} \\
& = & \mathsf{proj}(\forall\,(Q')\ \tau[\tau_0/\alpha]) & \text{by Property 1.3.3.i (page 40)}
\end{array}$$

This is the expected result

Property ii: Assume $\tau \equiv \tau'$. By definition, this means that we have $(\emptyset)\ \tau \equiv \tau'$. Hence, by Property i, we have $\tau/ = \tau'/$, that is $\mathsf{proj}(\tau)/ = \mathsf{proj}(\tau')/$. Hence, the skeletons $\mathsf{proj}(\tau)$ and $\mathsf{proj}(\tau')$ are equal. Since by definition $\mathsf{proj}(\tau)$ is τ and $\mathsf{proj}(\tau')$ is τ', we have the expected result, that is, $\tau = \tau'$.

Property iii: It is a direct consequence of property i. $\blacksquare$

1.5.3 Canonical forms for types

In this section, we define a function nf that maps all equivalent types to the same normal form, up to rearrangement.

Definition 1.5.5 We define the function $\mathsf{nf}(\sigma)$ inductively as follows:

$$\mathsf{nf}(\bot) \triangleq \bot \qquad \mathsf{nf}(\tau) \triangleq \tau \qquad \mathsf{nf}(\forall\,(\alpha \diamond \sigma)\ \sigma') \triangleq \begin{cases} \mathsf{nf}(\sigma) & \text{if } \mathsf{nf}(\sigma') \text{ is } \alpha \\ \mathsf{nf}(\sigma')[\tau/\alpha] & \text{if } \mathsf{nf}(\sigma) \text{ is } \tau \\ \mathsf{nf}(\sigma') & \text{if } \alpha \notin \mathsf{ftv}(\sigma') \\ \forall\,(\alpha \diamond \mathsf{nf}(\sigma))\ \mathsf{nf}(\sigma') & \text{otherwise} \end{cases}$$

If $\mathsf{nf}(\sigma) = \sigma$, we say that σ is in normal form. $\square$

We state here some useful properties about normal forms.

Properties 1.5.6 *The following hold for any type σ and any substitution θ:*

i) $\mathsf{nf}(\sigma) \equiv \sigma$.
ii) $\mathsf{ftv}(\sigma) = \mathsf{ftv}(\mathsf{nf}(\sigma))$.
iii) $\mathsf{nf}(\theta(\sigma)) = \theta(\mathsf{nf}(\sigma))$.
iv) $\mathsf{nf}(\sigma)$ *is in normal form.*

These properties are shown by induction on the number of universal quantifiers appearing in σ. The full proof can be found in Appendix (page 231).

In ML$^{\mathrm{F}}$, we distinguish monotypes τ and polytypes σ. Actually, we also need to spot *monotypes up to equivalence*, that is, polytypes that are equivalent to monotypes. Similarly, we spot *type variables up to equivalence*, that is, polytypes that are equivalent to type variables.

Definition 1.5.7 The set of polytypes σ such that $\mathsf{nf}(\sigma) = \tau$ for some monotype τ is written $\mathcal{T}$. The set of polytypes σ such that $\mathsf{nf}(\sigma) = \alpha$ for some type variable α is written $\mathcal{V}$. $\square$

By Rule EQ-MONO*, R-TRANS, and Property 1.5.6.i, if σ is in $\mathcal{T}$, we have $\forall\,(\alpha = \sigma)\ \sigma' \equiv \forall\,(\alpha \geq \sigma)\ \sigma'$. This is the reason why, in the following, any binding $(\alpha \diamond \sigma)$ where σ is in $\mathcal{T}$ is considered both flexible and rigid. In other words, flexibility is not meaningful for a bound in $\mathcal{T}$. This is not surprising since monotypes cannot be instantiated anyway.

Definition 1.5.8 The substitution extracted from a prefix Q, written $\widehat{Q}$, is defined by:

$$\widehat{\emptyset} \triangleq id \qquad\qquad \widehat{(\alpha \diamond \sigma, Q')} \triangleq \begin{cases} [\mathsf{nf}(\sigma)/\alpha] \circ \widehat{Q'} & \text{if } \sigma \in \mathcal{T} \\ \widehat{Q'} & \text{otherwise} \end{cases}$$

□

Notice that $\widehat{Q_1 Q_2} = \widehat{Q_1} \circ \widehat{Q_2}$ holds for any well-formed prefix $Q_1 Q_2$. Besides, for any σ and Q, we have $(Q)\ \sigma \equiv \widehat{Q}(\sigma)$ by using EQ-MONO repeatedly. Note also that the domain of $\widehat{Q}$ is the set of variables of $\mathsf{dom}(Q)$ whose bound is in $\mathcal{T}$. Therefore, $\mathsf{dom}(\widehat{Q}) \subseteq \mathsf{dom}(Q)$ holds. This can be compared with the domain of Θ_Q, which is always $\mathsf{dom}(Q)$.

As expected, normal forms capture the equivalence relation, up to commutation and alpha-conversion of binders. Since the equivalence relation is defined under a prefix, it has to be taken into account when comparing normal forms. This is stated precisely by the following lemma:

Lemma 1.5.9 *If* $(Q)\ \sigma_1 \equiv \sigma_2$ *holds, then so does* $\widehat{Q}(\mathsf{nf}(\sigma_1)) \approx \widehat{Q}(\mathsf{nf}(\sigma_2))$.

Proof: by induction on the derivation of $(Q)\ \sigma_1 \equiv \sigma_2$.

○ CASE EQ-REFL: Immediate.

○ CASE R-TRANS: By induction hypothesis and transitivity of the relation $\approx$.

○ CASE EQ-FREE: We have $\sigma_2 = \forall\,(\alpha \diamond \sigma)\ \sigma_1$ and $\alpha \notin \mathsf{ftv}(\sigma_1)$. By definition, $\mathsf{nf}(\sigma_2)$ is $\mathsf{nf}(\sigma_1)$, thus $\widehat{Q}(\mathsf{nf}(\sigma_1))$ is $\widehat{Q}(\mathsf{nf}(\sigma_2))$, and $\widehat{Q}(\mathsf{nf}(\sigma_1)) \approx \widehat{Q}(\mathsf{nf}(\sigma_2))$ holds.

○ CASE EQ-COMM: We have $\sigma_1 \approx \sigma_2$ by hypothesis, thus $\widehat{Q}(\mathsf{nf}(\sigma_1)) \approx \widehat{Q}(\mathsf{nf}(\sigma_2))$ holds by Property 1.5.3.ii.

○ CASE EQ-VAR: We have $\sigma_2 = \forall\,(\alpha \diamond \sigma_1)\ \alpha$. By definition, $\mathsf{nf}(\sigma_2)$ is $\mathsf{nf}(\sigma_1)$, thus we have $\widehat{Q}(\mathsf{nf}(\sigma_1)) = \widehat{Q}(\mathsf{nf}(\sigma_2))$, which implies $\widehat{Q}(\mathsf{nf}(\sigma_1)) \approx \widehat{Q}(\mathsf{nf}(\sigma_2))$.

○ CASE R-CONTEXT-R: We have $\sigma_1 = \forall\,(\alpha \diamond \sigma)\ \sigma_1'$ and $\sigma_2 = \forall\,(\alpha \diamond \sigma)\ \sigma_2'$. The premise is $(Q, \alpha \diamond \sigma)\ \sigma_1' \equiv \sigma_2'$ $(\underline{\mathbf{1}})$. By induction hypothesis and (1), we have the rearrangement $\widehat{(Q, \alpha \diamond \sigma)}(\mathsf{nf}(\sigma_1')) \approx \widehat{(Q, \alpha \diamond \sigma)}(\mathsf{nf}(\sigma_2'))$ $(\underline{\mathbf{2}})$. Note that $(Q, \alpha \diamond \sigma)$ is well-formed only if $\alpha \notin \mathsf{dom}(Q)$, thus, $\alpha \notin \mathsf{dom}(\widehat{Q}) \cup \mathsf{codom}(\widehat{Q})$ $(\underline{\mathbf{3}})$. We proceed by case analysis.

SUBCASE $\mathsf{nf}(\sigma)$ is τ: Let θ be $[\tau/\alpha]$. We have $\widehat{Q, \alpha \diamond \sigma} = \widehat{Q} \circ \theta$, thus, by (2), $\widehat{Q} \circ \theta(\mathsf{nf}(\sigma_1')) \approx \widehat{Q} \circ \theta(\mathsf{nf}(\sigma_2'))$. Since $\mathsf{nf}(\forall\,(\alpha \diamond \sigma)\ \sigma_1')$ is by definition $\theta(\mathsf{nf}(\sigma_1'))$ and $\mathsf{nf}(\forall\,(\alpha \diamond \sigma)$

σ'_2) is $\theta(\mathsf{nf}(\sigma'_2))$, we have shown that $\widehat{Q}(\mathsf{nf}(\sigma_1)) \approx \widehat{Q}(\mathsf{nf}(\sigma_2))$. In the following subcases, we assume that $\mathsf{nf}(\sigma)$ is not a monotype τ, which implies $\widehat{Q, \alpha \diamond \sigma} = \widehat{Q}$ **(4)**. Note also that $\widehat{Q}(\alpha) = \alpha$ **(5)** holds from (3). We have $\widehat{Q}(\mathsf{nf}(\sigma'_1)) \approx \widehat{Q}(\mathsf{nf}(\sigma'_2))$ **(6)** from (4) and (2).

Subcase $\mathsf{nf}(\sigma'_1)$ is α: From (6) and (5), we have $\alpha \approx \widehat{Q}(\mathsf{nf}(\sigma'_2))$. By Property 1.5.3.i, this gives $\widehat{Q}(\mathsf{nf}(\sigma'_2)) = \alpha$. Hence, $\mathsf{nf}(\sigma'_2)$ is α by (3). By definition, $\mathsf{nf}(\sigma_1)$ and $\mathsf{nf}(\sigma_2)$ are $\mathsf{nf}(\sigma)$, thus $\widehat{Q}(\mathsf{nf}(\sigma_1)) \approx \widehat{Q}(\mathsf{nf}(\sigma_2))$ holds by reflexivity.

Subcase $\mathsf{nf}(\sigma'_2)$ is α is similar.

Subcase $\alpha \notin \mathsf{ftv}(\mathsf{nf}(\sigma'_1))$: By (3), we get $\alpha \notin \mathsf{ftv}(\widehat{Q}(\mathsf{nf}(\sigma'_1)))$. By (6) and Property 1.5.4.iii, we get $\alpha \notin \mathsf{ftv}(\widehat{Q}(\mathsf{nf}(\sigma'_2)))$. By (3), we get $\alpha \notin \mathsf{ftv}(\mathsf{nf}(\sigma'_2))$. Hence, $\mathsf{nf}(\sigma_1)$ is $\mathsf{nf}(\sigma'_1)$ and $\mathsf{nf}(\sigma_2)$ is $\mathsf{nf}(\sigma'_2)$, thus we get the result by (6).

Subcase $\alpha \notin \mathsf{ftv}(\mathsf{nf}(\sigma'_2))$ is similar.

Otherwise, $\mathsf{nf}(\sigma)$ is not equal to a monotype τ, $\alpha \in \mathsf{ftv}(\mathsf{nf}(\sigma'_1))$, $\alpha \in \mathsf{ftv}(\mathsf{nf}(\sigma'_2))$, $\mathsf{nf}(\sigma'_1)$ is not α, and $\mathsf{nf}(\sigma'_2)$ is not α. By definition $\mathsf{nf}(\sigma_1)$ is $\forall\,(\alpha \diamond \mathsf{nf}(\sigma))\ \mathsf{nf}(\sigma'_1)$ and $\mathsf{nf}(\sigma_2)$ is $\forall\,(\alpha \diamond \mathsf{nf}(\sigma))\ \mathsf{nf}(\sigma'_2)$. From (6), we get $\forall\,(\alpha \diamond \widehat{Q}(\sigma))\ \widehat{Q}(\mathsf{nf}(\sigma'_1)) \approx \forall\,(\alpha \diamond \widehat{Q}(\sigma))\ \widehat{Q}(\mathsf{nf}(\sigma'_2))$, that is $\widehat{Q}(\mathsf{nf}(\sigma_1)) \approx \widehat{Q}(\mathsf{nf}(\sigma_2))$.

◦ Case R-Context-L: We have $\sigma_1 = \forall\,(\alpha \diamond \sigma'_1)\ \sigma$ and $\sigma_2 = \forall\,(\alpha \diamond \sigma'_2)\ \sigma$. The premise is $(Q)\ \sigma'_1 \equiv \sigma'_2$. By induction hypothesis, we have $\widehat{Q}(\mathsf{nf}(\sigma'_1)) \approx \widehat{Q}(\mathsf{nf}(\sigma'_2))$ **(7)**. We proceed by case analysis:

Subcase $\mathsf{nf}(\sigma)$ is α: Then $\mathsf{nf}(\sigma_1)$ is $\mathsf{nf}(\sigma'_1)$ and $\mathsf{nf}(\sigma_2)$ is $\mathsf{nf}(\sigma'_2)$, thus (7) is the expected result.

Subcase $\alpha \notin \mathsf{nf}(\sigma)$: $\mathsf{nf}(\sigma_1)$ is $\mathsf{nf}(\sigma)$ and $\mathsf{nf}(\sigma_2)$ is $\mathsf{nf}(\sigma)$. Hence, $\widehat{Q}(\mathsf{nf}(\sigma_1)) = \widehat{Q}(\mathsf{nf}(\sigma_2))$, which implies the expected result.

Subcase $\mathsf{nf}(\sigma'_1)$ is τ_1: By induction hypothesis (7), $\mathsf{nf}(\sigma'_2)$ is a monotype τ_2 such that $\widehat{Q}(\tau_1) = \widehat{Q}(\tau_2)$ **(8)** holds. By definition 1.5.5, $\mathsf{nf}(\sigma_1)$ is $\mathsf{nf}(\sigma)[\tau_1/\alpha]$ **(9)** and $\mathsf{nf}(\sigma_2)$ is $\mathsf{nf}(\sigma)[\tau_2/\alpha]$ **(10)**. Hence, we have:

$$\begin{array}{lcll}
\widehat{Q}(\mathsf{nf}(\sigma_1)) & = & \widehat{Q}(\mathsf{nf}(\sigma)[\tau_1/\alpha]) & \text{by (9)} \\
& = & \widehat{Q}(\mathsf{nf}(\sigma)[\widehat{Q}(\tau_1)/\alpha]) & \\
& = & \widehat{Q}(\mathsf{nf}(\sigma)[\widehat{Q}(\tau_2)/\alpha]) & \text{by (8)} \\
& = & \widehat{Q}(\mathsf{nf}(\sigma)[\tau_2/\alpha]) & \\
& = & \widehat{Q}(\mathsf{nf}(\sigma_2)) & \text{by (10)}
\end{array}$$

This is the expected result.

Subcase $\mathsf{nf}(\sigma'_2)$ is τ_2 is similar.

Otherwise, $\mathsf{nf}(\sigma_1)$ is $\forall\,(\alpha \diamond \mathsf{nf}(\sigma'_1))\ \mathsf{nf}(\sigma)$ and $\mathsf{nf}(\sigma_2)$ is $\forall\,(\alpha \diamond \mathsf{nf}(\sigma'_2))\ \mathsf{nf}(\sigma)$. Hence, we have $\widehat{Q}(\mathsf{nf}(\sigma_1)) \approx \widehat{Q}(\mathsf{nf}(\sigma_2))$ by (7).

◦ Case Eq-Mono: We have $\sigma_1 = \tau$, $\sigma_2 = \tau[\tau_0/\alpha]$, $(\alpha \diamond \sigma_0) \in Q$ **(11)**, and $(Q)\ \sigma_0 \equiv \tau_0$. By induction hypothesis, we have $\widehat{Q}(\mathsf{nf}(\sigma_0)) \approx \widehat{Q}(\tau_0)$, which implies $\widehat{Q}(\mathsf{nf}(\sigma_0)) = \widehat{Q}(\tau_0)$

by Property 1.5.3.i. Hence, $\mathsf{nf}(\sigma_0) = \tau'$ **(12)** such that $\widehat{Q}(\tau') = \widehat{Q}(\tau_0)$ **(13)**. We have $\widehat{Q}(\sigma_1) = \widehat{Q}(\tau) = \widehat{Q}(\tau[\tau'/\alpha])$ **(14)** by (11), by definition of $\widehat{Q}$, and by (12). We also have $\widehat{Q}(\sigma_2) = \widehat{Q}(\tau[\tau_0/\alpha]) = \widehat{Q}(\tau[\tau'/\alpha])$ **(15)** by (13). Hence, $\widehat{Q}(\sigma_1) = \widehat{Q}(\sigma_2)$ holds from (14) and (15), that is, $\widehat{Q}(\mathsf{nf}(\sigma_1)) = \widehat{Q}(\mathsf{nf}(\sigma_2))$ since $\mathsf{nf}(\sigma_1) = \mathsf{nf}(\tau) = \tau = \sigma_1$ and $\mathsf{nf}(\sigma_2) = \sigma_2$. ■

The next corollary characterizes equivalence under prefix. It shows that the only information that is used from the prefix Q is its associated substitution $\widehat{Q}$.

Corollary 1.5.10 *We have* $(Q)\ \sigma_1 \equiv \sigma_2$ *iff* $\widehat{Q}(\sigma_1) \equiv \widehat{Q}(\sigma_2)$.

See proof in the Appendix (page 232).

As a consequence, we have the following characterization of normal forms:

> $\mathsf{nf}(\sigma)$ is a rearrangement of $\mathsf{nf}(\sigma')$ iff $(Q)\ \sigma \equiv \sigma'$ holds for any well-formed prefix Q such that $\mathsf{utv}(\sigma, \sigma') \subseteq \mathsf{dom}(Q)$.

Next is a set of properties used throughout the thesis. For instance, Property i states that equivalent types have equal normal form, up to rearrangement. Property v shows that equivalence is preserved under substitution. Property vii shows that equivalent *monotypes* are actually equal, up to the substitution extracted from the prefix. Property viii will be used directly for showing the soundness of the unification algorithm. It states that two types are equivalent (under a given prefix) if and only if their subparts are equivalent.

Properties 1.5.11 *For any polytypes* σ, σ_1, σ_2, *any well-formed prefix* Q, *and any substitution* θ, *we have:*

i) *If* $\sigma_1 \equiv \sigma_2$, *then* $\mathsf{nf}(\sigma_1) \approx \mathsf{nf}(\sigma_2)$.
ii) σ *is in* $\mathcal{T}$ *iff* $\sigma \equiv \tau$ *for some monotype* τ
iii) σ *is in* $\mathcal{V}$ *iff* $\sigma \equiv \alpha$ *for some type variable* α
iv) $\theta(\sigma)$ *is in* $\mathcal{T}$ *iff* σ *is in* $\mathcal{T}$.
v) *If* $\sigma_1 \equiv \sigma_2$, *then* $\theta(\sigma_1) \equiv \theta(\sigma_2)$.
vi) *If* $(Q)\ \sigma_1 \equiv \sigma_2$, *then* $\widehat{Q}(\sigma_1)/ = \widehat{Q}(\sigma_2)/$.
vii) $(Q)\ \tau_1 \equiv \tau_2$ *iff* $\widehat{Q}(\tau_1) = \widehat{Q}(\tau_2)$.
viii) $(Q)\ g\ \tau_1 \ldots\ \tau_n \equiv g'\ \tau'_1 \ldots\ \tau'_m$ *holds if and only if* $g = g'$, $n = m$ *and* $(Q)\ \tau_i \equiv \tau'_i$ *holds for all* i *in* $1..n$.
ix) *If* $(Q)\ \sigma_1 \equiv \sigma_2$ *and* $\alpha \notin \mathsf{dom}(\widehat{Q})$ *hold, then* $\alpha \in \mathsf{ftv}(\sigma_1)$ *iff* $\alpha \in \mathsf{ftv}(\sigma_2)$.
x) *If* $\sigma_1 \in \mathcal{T}$ *and* $(Q)\ \sigma_1 \equiv \sigma_2$ *hold, then* $\sigma_2 \in \mathcal{T}$.

These properties are shown using Lemma 1.5.9 or Corollary 1.5.10.

Constructed forms

Types of the form $\forall\,(\alpha \diamond \sigma)\ \alpha$ are equivalent to σ by Rule EQ-VAR. However such types hide their structure in the bound of α. We sometimes need to force the structure of a type to be apparent. Hence, the following definition:

Definition 1.5.12 A type σ is in constructed form if and only if σ is $\bot$, or σ is of the form $\forall\,(Q)\ \tau$ and $\tau \notin \mathsf{dom}(Q)$. □

We note that every type τ is in constructed form. Moreover, every type admits a constructed form: the function cf, defined below, maps every type to a constructed form.

$$\mathsf{cf}(\bot) \triangleq \bot \qquad \mathsf{cf}(\tau) \triangleq \tau \qquad \mathsf{cf}(\forall\,(\alpha \diamond \sigma)\ \sigma') \triangleq \begin{cases} \mathsf{cf}(\sigma) & \text{if } \mathsf{nf}(\sigma') \text{ is } \alpha \\ \forall\,(\alpha \diamond \sigma)\ \mathsf{cf}(\sigma') & \text{otherwise} \end{cases}$$

Properties 1.5.13

i) *For any type σ, we have $\mathsf{cf}(\sigma) \equiv \sigma$.*

ii) *If σ is in constructed form, then $\mathsf{cf}(\sigma)$ is σ.*

The equivalence under (a given) prefix is a symmetric operation. In other words, it captures reversible transformations. Irreversible transformations are captured by an *instance* relation $\sqsubseteq$. Moreover, as explained in the introduction, we distinguish a subrelation $\sqsubseteq\!\!\!\!-$ of $\sqsubseteq$ called *abstraction*. The inverse of abstraction is used by type annotations to reveal the polymorphic type bound to a type variable in the prefix. In contrast, inverse of instance relations would, in general, be unsound. Indeed, reversing the instance relation is not sound in ML either.

1.6 The abstraction relation

Definition 1.6.1 (Abstraction) The *abstraction relation* $\sqsubseteq\!\!\!\!-$ is the smallest transitive and rigid-compliant relation under prefix that satisfies the rules of Figure 1.2. □

We write $(Q)\ \sigma_1 \sqsubseteq\!\!\!\!-\ \sigma_2$, which is read "$\sigma_2$ is an abstraction of σ_1" or "σ_1 is a revelation of σ_2" under prefix Q. The abstraction relation is a rigid-compliant relation under prefix, which means that abstraction under flexible bounds is not allowed. Equivalence is included in abstraction by Rule A-EQUIV. Rule A-HYP replaces a polytype σ_1 by a variable α_1, provided α_1 is rigidly bound to σ_1 in Q. Note that Rule A-HYP is not reversible. In particular, $(\alpha_1 = \sigma_1) \in Q$ does not imply $(Q)\ \alpha_1 \sqsubseteq\!\!\!\!-\ \sigma_1$, unless σ_1 is in $\mathcal{T}$. This asymmetry is essential, since uses of $\sqsubseteq\!\!\!\!-$ will be inferred, but uses of $\sqsupseteq\!\!\!\!-$ will

Figure 1.2: The abstraction relation

A-Equiv

$$\frac{(Q)\ \sigma_1 \equiv \sigma_2}{(Q)\ \sigma_1 \mathrel{-\!\!\!\!\sqsubseteq} \sigma_2}$$

A-Hyp

$$\frac{(\alpha_1 = \sigma_1) \in Q}{(Q)\ \sigma_1 \mathrel{-\!\!\!\!\sqsubseteq} \alpha_1}$$

not. When $(Q)\ \sigma_1 \mathrel{-\!\!\!\!\sqsubseteq} \alpha_1$ holds, we say that the polytype σ_1 is *abstracted* with α_1. This operation can also be seen as *hiding the polymorphic type* σ_1 *under the abstract name* α_1. Conversely, $(Q)\ \alpha_1 \mathrel{\sqsupseteq\!\!\!\!-} \sigma_1$ consists of *revealing* the polytype abstracted by the name α_1. An abstract polytype, *i.e.* a variable bound to a polytype in Q, can only be manipulated by its name, *i.e.* abstractly. The polytype must be revealed *explicitly* (by using the relation $\mathrel{\sqsupseteq\!\!\!\!-}$) before it can be further instantiated (along the relations $\mathrel{-\!\!\!\!\sqsubseteq}$ or $\sqsubseteq$). (See also Examples 6.2.12 and 6.2.13 below.)

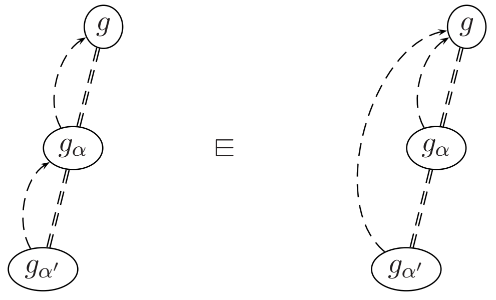
Example 1.6.3 The abstraction $(\alpha = \sigma)\ \forall(\alpha = \sigma)\ \sigma' \mathrel{-\!\!\!\!\sqsubseteq} \sigma'$ is derivable: on the one hand, $(\alpha = \sigma)\ \sigma \mathrel{-\!\!\!\!\sqsubseteq} \alpha$ holds by A-Hyp, leading to $(\alpha = \sigma)\ \forall(\alpha = \sigma)\ \sigma' \mathrel{-\!\!\!\!\sqsubseteq} \forall(\alpha = \alpha)\ \sigma'$ by R-Context-Rigid; on the other hand, $(\alpha = \sigma)\ \forall(\alpha = \alpha)\ \sigma' \equiv \sigma'$ holds by Eq-Mono*. Hence, we conclude by A-Equiv and R-Trans.

The following rule is derivable by A-Hyp, R-Context-Rigid and Eq-Mono*:

A-Up*

$$\frac{\alpha' \notin \mathsf{ftv}(\sigma_0)}{\forall(\alpha = \forall(\alpha' = \sigma')\ \sigma)\ \sigma_0 \mathrel{-\!\!\!\!\sqsubseteq} \forall(\alpha' = \sigma')\ \forall(\alpha = \sigma)\ \sigma_0}$$

This rule can be represented with graphs as follows:



The top level node is labelled with g, which is defined as σ_0/ϵ. The two other nodes are labelled with g_α and $g_{\alpha'}$, which are defined respectively as σ/ϵ and σ'/ϵ. The symbolic structure of the type is not known in detail. However, we expect α to be free in σ_0 and α' to be free in σ. This is represented by the doubled dashed lines.

Intuitively, only rigid bindings in Q are read (abstracted) in the derivation of $(Q)\ \sigma_1 \sqsubseteq\!\!\!\!-\ \sigma_2$, flexible bindings are simply ignored. Hence, if Q is unconstrained, it can be replaced by any other well-formed prefix, as stated by the following property:

Lemma 1.6.2 *If we have $\sigma_1 \sqsubseteq\!\!\!\!-\ \sigma_2$, then $(Q)\ \sigma_1 \sqsubseteq\!\!\!\!-\ \sigma_2$ holds whenever it is well-formed.*

Proof: This lemma is a consequence of the following, more general statement, taking $Q' = \emptyset$.

> *If Q is unconstrained and $(QQ')\ \sigma_1 \sqsubseteq\!\!\!\!-\ \sigma_2$ holds, if $\mathsf{dom}(Q) \subseteq \mathsf{dom}(Q'')$ holds and $Q''Q'$ is well-formed, then $(Q''Q')\ \sigma_1 \sqsubseteq\!\!\!\!-\ \sigma_2$ holds too.*

It is shown by induction on the derivation of $(QQ')\ \sigma_1 \sqsubseteq\!\!\!\!-\ \sigma_2$. Cases R-Trans, R-Context-R, and R-Context-Rigid are by induction hypothesis. Note that the reinforcement of the statement is necessary for the case R-Context-R.

∘ Case A-Equiv: By hypothesis $(QQ')\ \sigma_1 \equiv \sigma_2$ holds. Let θ be $\widehat{Q} \circ \widehat{Q'}$. By Corollary 1.5.10, we have $\theta(\sigma_1) \equiv \theta(\sigma_2)$ (**1**) Since Q is unconstrained, $\widehat{Q}$ is the identity. Hence, θ is $\widehat{Q'}$. As a consequence, $\widehat{Q''Q'}$ is $\widehat{Q''} \circ \theta$ (**2**). By Property 1.5.11.v and (1), we have $\widehat{Q''} \circ \theta(\sigma_1) \equiv \widehat{Q''} \circ \theta(\sigma_2)$. By Corollary 1.5.10 and (2), we get $(Q''Q')\ \sigma_1 \equiv \sigma_2$. Hence, $(Q''Q')\ \sigma_1 \sqsubseteq\!\!\!\!-\ \sigma_2$ holds by A-Equiv.

∘ Case A-Hyp: We have $\sigma_2 = \alpha$ and $(\alpha = \sigma_1) \in QQ'$. Since Q is unconstrained, we must have $(\alpha = \sigma_1) \in Q'$. Hence, $(\alpha = \sigma_1) \in Q''Q'$, thus $(Q''Q')\ \sigma_1 \sqsubseteq\!\!\!\!-\ \alpha$ holds by A-Hyp. This is the expected result. ■

The abstraction relation can only hide a polytype under its abstract name, but cannot modify the structure of the polytype. This is why this relation is (explicitly) reversible. In order to instantiate polytypes, as we do in ML, we need an irreversible instance relation.

1.7 The instance relation

Definition 1.7.1 (Instance) The *instance relation* $\sqsubseteq$ is the smallest transitive and flexible-compliant relation under prefix that satisfies the rules of Figure 1.3. □

We write $(Q)\ \sigma_1 \sqsubseteq \sigma_2$, which is read "$\sigma_2$ is an instance of σ_1" or "σ_1 is more general than σ_2" under prefix Q.

It is a flexible-compliant relation under prefix, thus only flexible bindings may be instantiated. Conversely, instantiation cannot occur under rigid bounds, except when it is an abstraction, and we have to use I-Abstract then. Rule I-Bot means that $\bot$ behaves as a least element for the instance relation. Rule I-Rigid mean that flexible bounds can be changed into rigid bounds. An interesting rule is I-Hyp—the counterpart of rule A-Hyp, which replaces a polytype σ_1 by a variable α_1, provided σ_1 is a flexible bound of α_1 in Q.

Example 1.7.4 The instance relation $(\alpha \geq \sigma)\, \forall\, (\alpha \geq \sigma)\, \sigma' \sqsubseteq \sigma'$ holds. The derivation follows the one of Example 1.6.3 but uses I-Hyp and R-Context-Flexible instead of A-Hyp and R-Context-Rigid. More generally, the following rule is derivable:

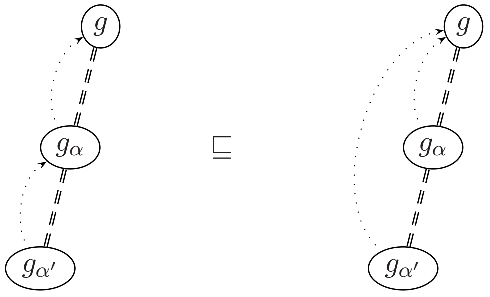
$$\text{I-DROP}^\star \qquad \frac{}{(QQ'Q'')\, \forall\, (Q')\, \sigma \sqsubseteq \sigma}$$

Example 1.7.5 The relation[1] $(Q)\, \forall\, (\alpha \geq \sigma)\, \forall\, (\beta \geq \sigma)\, \alpha \to \beta \sqsubseteq \forall\, (\alpha \geq \sigma)\, \forall\, (\beta \geq \alpha)\, \alpha \to \beta$ follows by rules R-Context-R, R-Context-Flexible, and I-Hyp. The right hand side is equivalent to $\forall\, (\alpha \geq \sigma)\, \alpha \to \alpha$ by Eq-Mono*.

The following rules are derivable as well:

$$\text{I-EQUIV}^\star \quad \frac{(Q)\, \sigma_1 \equiv \sigma_2}{(Q)\, \sigma_1 \sqsubseteq \sigma_2} \qquad \text{I-UP}^\star \quad \frac{\alpha_2 \notin \mathsf{ftv}(\sigma)}{(Q)\, \forall\, (\alpha_1 \geq \forall\, (\alpha_2 \diamond \sigma_2)\, \sigma_1)\, \sigma \sqsubseteq \forall\, (\alpha_2 \diamond \sigma_2)\, \forall\, (\alpha_1 \geq \sigma_1)\, \sigma}$$

We represent I-Up* with graphs as follows:



We now state a few simple properties about abstraction and instantiation, using the meta-symbol $\diamondsuit$.

---

[1] The single difference between the two types is highlighted.

Figure 1.3: Type instance

$$\textsc{I-Abstract}\ \frac{(Q)\ \sigma_1 \mathrel{-\mkern-14mu\sqsubseteq} \sigma_2}{(Q)\ \sigma_1 \sqsubseteq \sigma_2} \qquad \textsc{I-Hyp}\ \frac{(\alpha_1 \geq \sigma_1) \in Q}{(Q)\ \sigma_1 \sqsubseteq \alpha_1} \qquad \textsc{I-Bot}\ (Q)\ \bot \sqsubseteq \sigma \qquad \textsc{I-Rigid}\ \frac{}{(Q)\ \forall\,(\alpha \geq \sigma_1)\ \sigma \sqsubseteq \forall\,(\alpha = \sigma_1)\ \sigma}$$

Properties 1.7.2

i) *Given a renaming ϕ of $\mathsf{dom}(Q)$, if $(Q)\ \sigma_1 \diamondsuit \sigma_2$ holds, then we have a derivation with the same size of $(\phi(Q))\ \phi(\sigma_1) \diamondsuit \phi(\sigma_2)$.*

ii) *If $Q' \approx Q$ and $(Q)\ \sigma_1 \diamondsuit \sigma_2$ hold, then we have $(Q')\ \sigma_1 \diamondsuit \sigma_2$.*

iii) *If QQ' is well-formed and $(Q)\ \sigma_1 \diamondsuit \sigma_2$ holds, then we have $(QQ')\ \sigma_1 \diamondsuit \sigma_2$.*

The proofs are by induction on the input derivation.

1.8 Examples

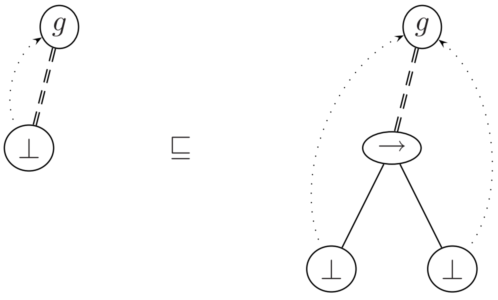
In this section, we consider two examples. The first one illustrates how the usual ML instance relation is subsumed by the instance relation of ML$^{\text{F}}$. The second example considers a well-known isomorphism in System F, which is partially captured by our instance relation.

Example 1.8.6 The relation $\sqsubseteq$ generalizes the instance relation of ML. For example, the type $\forall\,(\alpha)\ \tau[\alpha]$ is more general than $\forall\,(\alpha, \alpha')\ \tau[\alpha \to \alpha']$, as shown by the following derivation (valid under any prefix):

$$\begin{array}{ll}
\ \forall\,(\alpha)\ \tau[\alpha] & \\
= \forall\,(\alpha \geq \bot)\ \tau[\alpha] & (\underline{\mathbf{1}}) \\
\equiv \forall\,(\alpha', \alpha'')\ \forall\,(\alpha \geq \bot)\ \tau[\alpha] & (\underline{\mathbf{2}}) \\
\sqsubseteq \forall\,(\alpha', \alpha'')\ \forall\,(\alpha \geq \alpha' \to \alpha'')\ \tau[\alpha] & (\underline{\mathbf{3}}) \\
\equiv \forall\,(\alpha', \alpha'')\ \tau[\alpha' \to \alpha''] & (\underline{\mathbf{4}}) \\
= \forall\,(\alpha, \alpha')\ \tau[\alpha \to \alpha'] & (\underline{\mathbf{5}})
\end{array}$$

(1) by notation; (2) by Eq-Free; (3) by R-Context-Flexible, since $(\alpha', \alpha'')\ \bot \sqsubseteq \alpha' \to \alpha''$; (4) by Eq-Mono*; and (5) by renaming.

We represent this instance relation by the graphs below. The symbol g is the top symbol of τ, that is, τ/ϵ. The node labelled with $\bot$ in the left-hand graph represents α. The two nodes labelled with $\bot$ in the right-hand graph represent α and α', respectively.

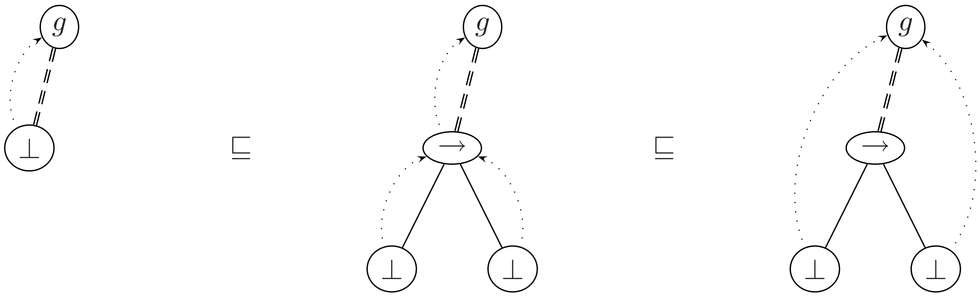


Note that another derivation is possible:

$$\begin{array}{cll}
 & \forall\,(\alpha)\ \tau[\alpha] & \\
= & \forall\,(\alpha \geq \bot)\ \tau[\alpha] & (\underline{\mathbf{6}}) \\
\sqsubseteq & \forall\,(\alpha \geq \forall\,(\alpha', \alpha'')\ \alpha' \to \alpha'')\ \tau[\alpha] & (\underline{\mathbf{7}}) \\
\sqsubseteq & \forall\,(\alpha', \alpha'')\ \forall\,(\alpha \geq \alpha' \to \alpha'')\ \tau[\alpha] & (\underline{\mathbf{8}}) \\
\equiv & \forall\,(\alpha', \alpha'')\ \tau[\alpha' \to \alpha''] & (\underline{\mathbf{9}}) \\
\equiv & \forall\,(\alpha, \alpha')\ \tau[\alpha \to \alpha'] & (\underline{\mathbf{10}})
\end{array}$$

(6) by notation; (7) by R-Context-Flexible and I-Bot; (8) by I-Up*; (9) by Eq-Mono*; and (10) by renaming.

Two instantiations are used (namely (7) and (8)). We represent them with the following graphs:



Example 1.8.7 The instance relation of ML$^{\mathrm{F}}$ covers an interesting case of type isomorphism [Cos95]. In System F, type $\forall\,\alpha \cdot \tau' \to \tau$ is isomorphic[2] to $\tau' \to \forall\,\alpha \cdot \tau$ whenever α is not free in τ'. In ML$^{\mathrm{F}}$, the two corresponding polytypes are not equivalent but in an instance relation. Precisely, $\forall\,(\alpha' \geq \forall\,(\alpha)\ \tau)\ \tau' \to \alpha'$ is more general than $\forall\,(\alpha)\ \tau' \to \tau$, as shown by the following derivation:

$$\begin{array}{ll}
\ \forall\,(\alpha' \geq \forall\,(\alpha)\ \tau)\ \tau' \to \alpha' & \\
\sqsubseteq \forall\,(\alpha)\ \forall\,(\alpha' \geq \tau)\ \tau' \to \alpha' & \text{by I-Up}^\star \\
\equiv \forall\,(\alpha)\ \tau' \to \tau & \text{by Eq-Mono}^\star
\end{array}$$

---

[2] That is, there exists a function $\eta\beta$-reducible to the identity that transforms one into the other, and conversely.

As a particular case, let $\sigma_{\mathtt{id}} \triangleq \forall\,(\beta)\ \beta \to \beta$; then $\forall\,(\alpha, \alpha' \geq \sigma_{\mathtt{id}})\ \alpha \to \alpha'$ is more general than $\forall\,(\alpha, \beta)\ \alpha \to \beta \to \beta$ (the derivation closely follows the one of example 1.8.6). It should be noticed that we encoded the System F type $\tau' \to \forall\,\alpha \cdot \tau$ by $\forall\,(\alpha' \geq \forall\,(\alpha)\ \tau)\ \tau' \to \alpha'$. Another valid encoding would be $\forall\,(\alpha' = \forall\,(\alpha' = \forall\,(\alpha)\ \tau))\ \tau' \to \alpha$. Then the bound of α' is rigid, and the derivation given above is not possible. In that case, the types are not in an instance relation. However, as in System F, it is possible to write a retyping function from one type to the other.

This example shows that ML^{F} types and instance relation can capture part of this isomorphism. This is actually a standard approach when addressing the problem of type inference with first class polymorphism. For example, the intermediate language Λ_2^- [KW94] forbids quantifiers at the right of arrows. This language is shown equivalent to System F, thanks to the aforementioned isomorphism. As another example, taken from Odersky and Läufer's article [OL96] $\forall\,\alpha \cdot \mathtt{int} \to \alpha\ \mathtt{list}$ is made a subtype of $\mathtt{int} \to \forall\,\alpha \cdot \alpha\ \mathtt{list}$ using an appropriate instance relation.

In summary, ML^{F} partially captures the type isomorphism given above. It should also be noted that, as opposed to type containment [Mit88], the instance relation cannot express any form of contravariance.

Chapter 2

Properties of relations under prefixes

Instantiation in ML consists of applying a substitution: given a type scheme $\sigma \triangleq \forall (\bar{\alpha})\ \tau$, its instances are of the form $\theta(\tau)$ for a substitution θ whose domain is included in $\bar{\alpha}$. This definition of ML instantiation implies some useful properties, which we expect to hold in ML$^{\mathrm{F}}$ too.

In Section 2.1, we establish a few properties based on the observation of the skeletons. In ML, a type σ and its instances have comparable structures, that is, comparable skeletons. More precisely, the skeleton of the instance $\theta(\tau)$ corresponds to the skeleton of σ where quantified variables $\bar{\alpha}$ are substituted by θ. Hence, the skeleton of $\theta(\tau)$ is the skeleton of σ, except on occurrences corresponding to quantified variables. In ML$^{\mathrm{F}}$, we define a partial order $\leqslant_{/}$ on occurrences (or, equivalently, on skeletons) that captures the idea that only quantified variables can be instantiated. This result is stated as Property 2.1.3.ii, which concerns the instance relation. Conversely, the abstraction relation is, intuitively, a reversible relation. Hence, we expect it to keep skeletons unchanged; this is stated by Property 2.1.3.i. Another property of the ML instance relation concerns free variables: if α is free in σ, then α is free in all instances of σ (and more precisely, α appears at least at the same occurrences). Such a result also holds in ML$^{\mathrm{F}}$, as stated by Lemma 2.1.4, which can be first read by taking $Q_1 = \emptyset$. A straightforward result in ML states that monotypes have no instances (except themselves). This is also true in ML$^{\mathrm{F}}$, up to equivalence, as stated by Lemma 2.1.6.

In some implementations of ML, such as OCaml, some type variables become aliases during unification. For example, the unification of $\alpha \to \alpha$ and $\beta \to \beta$ might create an alias from α to β, which we write $(\alpha = \beta)$ $(\underline{\mathbf{1}})$. Then each time a type variable is encountered, the implementation calls a function *repr* which finds its representative. To pursue the example, the representative of α is β and the representative of β is β.

Such a mechanism plays quite an important role in the efficiency of unification in ML. However, it is seldom formalized. In ML$^{\mathrm{F}}$, the alias from α to β appears directly in the prefix, in the form (1). In Section 2.2, we define the representative of a variable, according to a given prefix, and establish a few properties. The representative of α in Q is written $Q[\alpha]$, and the bound of the representative (which, intuitively, is the bound meant for α), is written $Q(\alpha)$.

Section 2.3 provides different ways to transform derivations of equivalence, abstraction, or instance judgments, the goal being to force some invariants. A first set of straightforward restrictions defines *restricted* derivations; another independent restriction, which concerns only the context rule R-Context-R, defines *thrifty* derivations. Property 2.3.3 shows that derivations that are restricted and thrifty are as expressive as normal derivations. Such restrictions are useful to discard pathological cases, which would shamelessly obfuscate some proofs.

In Section 2.4 we study contexts, which represent types with a hole [ ]. We use them in Section 2.5 to define a set of rules that replaces context rules by explicit contexts. More precisely, context rules such as R-Context-R, R-Context-Flexible or R-Context-Rigid are removed, and all other rules mention explicitly a context C. As expected, these new relations are as expressive as the original relations. Another independent set of rules is introduced, that defines the relations $\mathrel{\sqsubset\!\!\!\!-}^{\bar\alpha}$ and $\sqsubseteq^{\bar\alpha}$. These relations are similar to $\mathrel{\sqsubset\!\!\!\!-}$ and $\sqsubseteq$, but seriously restrict rules A-Hyp and I-Hyp. In order to preserve the expressiveness, we introduce new rules, namely A-Up', A-Alias', I-Up', and I-Alias'. The derivations with $\mathrel{\sqsubset\!\!\!\!-}^{\bar\alpha}$ and $\sqsubseteq^{\bar\alpha}$ make the transformations on binders more explicit than with $\mathrel{\sqsubset\!\!\!\!-}$ and $\sqsubseteq$. More precisely, whereas the latter only consider unification of binders (rules A-Hyp and I-Hyp), the former distinguish local unification of binders (rules A-Alias' and I-Alias'), extrusion of binders (rules A-Up' and I-Up'), and unification of binders with the prefix (rules A-Hyp' and I-Hyp').

In Section 2.6, we define *atomic* relations. Atomic relations can decompose an abstraction derivation or an instance derivation into a sequence of atomic and effective transformations. We use atomic decomposition to show some confluence results; indeed, once a derivation is decomposed into a sequence of transformations, confluence can be proved by considering all possible pairs of transformations.

The main result of Section 2.7 is Property 2.7.7.i, which shows that the equivalence relation is the symmetric kernel of the instance relation. This result is proved by associating a three-variable polynomial to each type, and by showing that the instance relation strictly decreases the polynomials if and only if it is irreversible (that is, when it is not an equivalence). Polynomials are also used to show that the relation $\mathrel{\sqsubset\!\!\!\!-}$ considered as an order on types (under a given prefix) is well-founded. Such a result is used in the following section to show confluence between $\mathrel{\sqsubset\!\!\!\!-}$ and $\sqsubseteq$. Indeed, the main result of Section 2.8 is the Diamond Lemma (Lemma 2.8.4), which states the confluence of $\mathrel{\sqsubset\!\!\!\!-}$ and $\sqsubseteq$ under an unconstrained prefix.

2.1 Projections and instantiation

Skeletons, projections and occurrences were defined formally in section 1.3. We have seen that a skeleton is a tree composed of type variables, type constructors g^n (such as the arrow $\rightarrow$), and $\bot$. Intuitively, only the leaves labeled with $\bot$ can be instantiated. This is captured by the following definition, which introduces the notation $\leqslant_{/}$. Then, we immediately show that this relation is actually a partial order on skeletons.

Definition 2.1.1 We define a relation on skeletons, written $\leqslant_{/}$, as follows: $t_1 \leqslant_{/} t_2$ holds if and only if $\mathsf{dom}(t_1) \subseteq \mathsf{dom}(t_2)$ and for all $u \in \mathsf{dom}(t_1)$, we have $t_1/u \neq t_2/u$ implies $t_1/u = \bot$. □

The definition immediately applies to projections since they are isomorphic to skeletons (see Section 1.3.2). By extension, we write $\sigma_1 \leqslant_{/} \sigma_2$ to mean $\sigma_1/ \leqslant_{/} \sigma_2/$.

Properties 2.1.2

i) *The relation $\leqslant_{/}$ is a partial order.*

ii) *If $t_1 \leqslant_{/} t_2$ holds, then for any substitution θ, we have $\theta(t_1) \leqslant_{/} \theta(t_2)$.*

iii) *If $t_1 \leqslant_{/} t_2$ holds, then for any skeleton t, we have $t[t_1/\alpha] \leqslant_{/} t[t_2/\alpha]$.*

These properties are direct consequences of Definition 2.1.1.
See details in the Appendix (page 233).

As a consequence of Property 2.1.2.i, if we have $\sigma_1 \leqslant_{/} \sigma_2$ and $\sigma_2 \leqslant_{/} \sigma_1$, then $\sigma_1/ = \sigma_2/$ (that is, $\mathsf{proj}(\sigma_1) = \mathsf{proj}(\sigma_2)$). However, this does not imply $\sigma_1 = \sigma_2$ in general, as shown by example 1.3.1. Actually, it does not imply $\sigma_1 \equiv \sigma_2$ either, since types contain more information than skeletons.

Projections, instantiation and free variables

Projections are stable under abstraction, but not (in general) under instantiation. This is stated more precisely by the following lemma:

Properties 2.1.3

i) *If $(Q)\ \sigma_1 \sqsubseteq\!\!\!\!\!\equiv \sigma_2$, then $(\forall\,(Q)\ \sigma_1)/ = (\forall\,(Q)\ \sigma_2)/$.*

ii) *If $(Q)\ \sigma_1 \sqsubseteq \sigma_2$, then $\forall\,(Q)\ \sigma_1 \leqslant_{/} \forall\,(Q)\ \sigma_2$.*

Proof: Property i : It is shown by induction on the derivation of $(Q)\ \sigma_1 \sqsubseteq\!\!\!\!\!\equiv \sigma_2$.

- ∘ Case A-Equiv: By Property 1.5.4.i (page 50).
- ∘ Case R-Trans: By induction hypothesis.

∘ Case A-Hyp: By hypothesis, we have $(\alpha = \sigma_1) \in Q$ and σ_2 is α. We have $\forall\,(Q)\;\sigma_2 = \forall\,(Q)\;\alpha \equiv \forall\,(Q)\;\sigma_1$ by Eq-Var*. Consequently, $\forall\,(Q)\;\sigma_1 \equiv \forall\,(Q)\;\sigma_2$ holds. We get the expected result by Property 1.5.4.i (page 50).

∘ Case R-Context-R: We have $\sigma_1 = \forall\,(\alpha \diamond \sigma)\;\sigma'_1$ and $\sigma_2 = \forall\,(\alpha \diamond \sigma)\;\sigma'_2$. Besides, the premise is $(Q, \alpha \diamond \sigma)\;\sigma'_1 \mathrel{-\!\!\!\!\sqsubset} \sigma'_2$. Hence, by induction hypothesis, we have $(\forall\,(Q, \alpha \diamond \sigma)\;\sigma'_1)/ = (\forall\,(Q, \alpha \diamond \sigma)\;\sigma'_2)/$, which is the expected result.

∘ Case R-Context-Rigid: Like for the case R-Context-L in the proof of Property 1.5.4.i (page 50).

Property ii : It is shown by induction on the derivation of $(Q)\;\sigma_1 \sqsubseteq \sigma_2$.

∘ Case I-Abstract: By Property i.

∘ Case R-Trans: By induction hypothesis and transitivity of $\leqslant_{/}$ (Property 2.1.2.i).

∘ Case I-Bot: We have $\sigma = \bot$, thus $(\forall\,(Q)\;\sigma_1)/ = \bot/$. We get the expected result by observing that $\bot \leqslant_{/} (\forall\,(Q)\;\sigma_2)$ always holds.

∘ Case R-Context-Flexible: We have $\sigma_1 = \forall\,(\alpha \geq \sigma'_1)\;\sigma$ $\mathbf{(\underline{1})}$ and $\sigma_2 = \forall\,(\alpha \geq \sigma'_2)\;\sigma$ $\mathbf{(\underline{2})}$. The premise is $(Q)\;\sigma'_1 \sqsubseteq \sigma'_2$. By induction hypothesis, we know that $(\forall\,(Q)\;\sigma'_1)/ \leqslant (\forall\,(Q)\;\sigma'_2)/$, that is $\Theta_Q(\mathsf{proj}(\sigma'_1))/ \leqslant \Theta_Q(\mathsf{proj}(\sigma'_2))/$ $\mathbf{(\underline{3})}$ by Property 1.3.3.i (page 40). We have

$$\begin{array}{rcll}
(\forall\,(Q)\;\sigma_1)/ & = & (\forall\,(Q)\;\forall\,(\alpha \geq \sigma'_1)\;\sigma)/ & \text{from (1)} \\
 & = & \Theta_Q(\mathsf{proj}(\sigma)[\quad \mathsf{proj}(\sigma'_1)/\alpha])/ & \text{by Property 1.3.3.i (page 40)} \\
 & & & \text{and Definition 1.3.2} \\
 & = & \Theta_Q(\mathsf{proj}(\sigma))[\Theta_Q(\mathsf{proj}(\sigma'_1))/\alpha]/ & \\
 & \leqslant_{/} & \Theta_Q(\mathsf{proj}(\sigma))[\Theta_Q(\mathsf{proj}(\sigma'_2))/\alpha]/ & \text{by Property 2.1.2.iii and (3).} \\
 & = & \Theta_Q(\mathsf{proj}(\sigma)[\quad \mathsf{proj}(\sigma'_2)/\alpha])/ & \\
 & = & (\forall\,(Q)\;\forall\,(\alpha \geq \sigma'_2)\;\sigma)/ & \text{by Property 1.3.3.i (page 40)} \\
 & & & \text{and Definition 1.3.2} \\
 & = & (\forall\,(Q)\;\sigma_2)/ & \text{from (2)}
\end{array}$$

Finally, we have shown $\forall\,(Q)\;\sigma_1/ \leqslant_{/} \forall\,(Q)\;\sigma_2/$, which is the expected result.

∘ Case I-Hyp is similar to case A-Hyp above.

∘ Case I-Rigid: $\forall\,(\alpha \geq \sigma)\;\sigma'$ and $\forall\,(\alpha = \sigma)\;\sigma'$ have the same projection. ■

Considering an instance $(Q)\;\sigma_1 \sqsubseteq \sigma_2$, we need to track occurrences of variables bound in Q but free in σ_1. The following lemma states that, in general, such variables also occur in σ_2, at the same occurrence. We require $(\alpha \diamond \sigma)$ to be in the prefix, because the instance relation is only defined under a prefix that binds all free variables of the judgment. Additionally, we require σ not to be equivalent to a monotype, otherwise it could be equivalently substituted by Rule Eq-Mono, and the result would not hold. As a counterexample, take $(Q, \alpha = \tau)\;\alpha \sqsubseteq \tau$ (where α cannot be free in τ).

Lemma 2.1.4 *Assume σ is not in $\mathcal{T}$. If $(Q, \alpha \diamond \sigma, Q_1)\ \sigma_1 \diamondsuit \sigma_2$ and $(\forall\,(Q_1)\ \sigma_1)/u = \alpha$ hold, then $(\forall\,(Q_1)\ \sigma_2)/u = \alpha$. As a consequence, if $\alpha \in \mathsf{ftv}(\forall\,(Q_1)\ \sigma_1)$, then $\alpha \in \mathsf{ftv}(\forall\,(Q_1)\ \sigma_2)$.*

The proof is by induction on the derivation. See the details in Appendix (page 234).

Monotypes

By Definition 1.5.7, a type σ is in $\mathcal{T}$ if its normal form has no quantifiers. Like in ML, this means that monotypes cannot be instantiated. Monotypes play an important role in ML^{F}, since they can be inferred (like in ML), whereas polymorphic types can only be propagated. Throughout the proofs, we need to characterize monotypes, variables, or $\bot$. Definition 1.5.7 uses normal forms. The following properties use projection.

Properties 2.1.5
i) *$\sigma \in \mathcal{T}$ iff for all $u \in \mathsf{dom}(\sigma)$ we have $\sigma/u \neq \bot$.*
ii) *We have $\sigma/\epsilon = \alpha$ iff $\sigma \equiv \alpha$.*
iii) *We have $\sigma/\epsilon = \bot$ iff $\sigma \equiv \bot$.*

See proof in the Appendix (page 235).

The following lemma is essential: it shows that any instance of a monotype τ is only equivalent to τ, even under a prefix. In particular, assume σ is some polymorphic type scheme. If we have $(\alpha = \sigma)\ \alpha \sqsubseteq \sigma'$, then σ' must be α (when put in normal form). In other words, even if α is rigidly bound to a polytype σ in the prefix, there is no way to take an instance of it other than α itself. This is also why we say that the information $(\alpha = \sigma)$ is hidden in the prefix: we can propagate the variable α, bound to σ, but we cannot use its polymorphism. Decidability of type inference relies on this result.

Lemma 2.1.6 *If $\sigma_1 \in \mathcal{T}$ and $(Q)\ \sigma_1 \diamondsuit \sigma_2$ hold, then we have $(Q)\ \sigma_1 \equiv \sigma_2$.*

Proof: By induction on the derivation of $(Q)\ \sigma_1 \diamondsuit \sigma_2$.

- Case A-Equiv: By hypothesis, $(Q)\ \sigma_1 \equiv \sigma_2$ holds, which is the expected result.
- Case R-Trans By induction hypothesis, Property 1.5.11.x (page 54) and R-Trans.
- Case A-Hyp and I-Hyp: We have $(Q)\ \sigma_1 \sqsubseteq \alpha_1$ (that is, σ_2 is α_1), and the premise is $(\alpha_1 \diamond \sigma_1) \in Q$ $\mathbf{(\underline{1})}$. By hypothesis, $\sigma_1 \in \mathcal{T}$, thus $\sigma_1 \equiv \tau$ $\mathbf{(\underline{2})}$ for some monotype τ by Property 1.5.11.ii (page 54). By Property 1.5.3.v (page 49) and (2), we get $(Q)\ \sigma_1 \equiv \tau$ $\mathbf{(\underline{3})}$. By (2), Eq-Mono, and (1), we get $(Q)\ \tau \equiv \alpha_1$ $\mathbf{(\underline{4})}$. Hence, $(Q)\ \sigma_1 \equiv \alpha_1$ holds by R-Trans, (3), and (4). This is the expected result.

∘ Case R-Context-R We have $\sigma_1 = \forall\,(\alpha \diamond \sigma)\ \sigma'_1$ and $\sigma_2 = \forall\,(\alpha \diamond \sigma)\ \sigma'_2$. The premise is $(Q, \alpha \diamond \sigma)\ \sigma'_1 \mathrel{\Diamond} \sigma'_2$ $\mathbf{(\underline{5})}$. By hypothesis we have $\forall\,(\alpha \diamond \sigma)\ \sigma'_1 \in \mathcal{T}$. By Property 2.1.5.i, we must have $\sigma'_1 \in \mathcal{T}$. Hence $(Q, \alpha \diamond \sigma)\ \sigma'_1 \equiv \sigma'_2$ holds by induction hypothesis on (5). Consequently, $(Q)\ \forall\,(\alpha \diamond \sigma)\ \sigma'_1 \equiv \forall\,(\alpha \diamond \sigma)\ \sigma'_2$ holds by R-Context-R and this is the expected result.

∘ Case R-Context-Rigid and R-Context-Flexible: We have $\sigma_1 = \forall\,(\alpha \diamond \sigma)\ \sigma_0$ and $\sigma_2 = \forall\,(\alpha \diamond \sigma')\ \sigma_0$. The premise is $(Q)\ \sigma \mathrel{\Diamond} \sigma'$ $\mathbf{(\underline{6})}$. If $\alpha \notin \mathsf{ftv}(\sigma_0)$, then $\sigma_1 \equiv \sigma_2$ by Eq-Free, which is the expected result. Otherwise, we necessarily have σ and σ_0 in $\mathcal{T}$ by Property 2.1.5.i. Hence, $(Q)\ \sigma \equiv \sigma'$ holds by induction hypothesis on (6). Consequently, $(Q)\ \sigma_1 \equiv \sigma_2$ holds by R-Context-L.

∘ Case I-Bot implies $\sigma_1 = \bot$, which is a contradiction with the hypothesis $\sigma_1 \in \mathcal{T}$, by Property 2.1.5.i.

∘ Case I-Rigid: we have $\sigma_1 = \forall\,(\alpha \geq \sigma)\ \sigma'$ and $\sigma_2 = \forall\,(\alpha = \sigma)\ \sigma'$. If $\alpha \notin \mathsf{ftv}(\sigma')$, then $\sigma_1 \equiv \sigma_2$. Otherwise, we necessarily have σ and σ' in $\mathcal{T}$ by Property 2.1.5.i. Consequently, $\sigma \equiv \tau$ for some monotype τ. Hence $\sigma_1 \equiv \sigma'[\tau/\alpha]$ $\mathbf{(\underline{7})}$ holds by Eq-Mono* as well as $\sigma'[\tau/\alpha] \equiv \sigma_2$ $\mathbf{(\underline{8})}$. We get the expected result by R-Trans, (7), (8), and by Property 1.5.3.v (page 49). ■

Cycles

Types in ML$^{\text{F}}$ are not recursive, thus a type cannot be strictly included in itself. As a consequence, an instance of a given polytype cannot be a strict subterm.

Properties 2.1.7

i) *If* $\sigma \cdot \epsilon/ \leqslant_{/} \sigma \cdot u/$, *then* u *is* ϵ.

ii) *If we have either* $(Q)\ \sigma \sqsubseteq \alpha$ *or* $(Q)\ \alpha \sqsubseteq \sigma$ *and if* $\alpha \in \mathsf{ftv}(\sigma)$, *then* $\sigma \equiv \alpha$.

Proof: Property i: We have $\sigma \cdot \epsilon/ \leqslant_{/} \sigma \cdot u/$. By definition, this implies $\sigma/u \leqslant_{/} \sigma/uu$. Hence uu, that is u^2, is in $\mathsf{dom}(\sigma)$. By immediate induction, we show that u^i is in $\mathsf{dom}(\sigma)$ for all natural number i. By definition, $\mathsf{dom}(\sigma)$ is $\mathsf{dom}(\mathsf{proj}(\sigma))$, where $\mathsf{proj}(\sigma)$ is a skeleton (that is, a tree). By construction, $\mathsf{dom}(\mathsf{proj}(\sigma))$ is a finite set of occurrences. Hence, we necessarily have $u = \epsilon$.

Property ii: By hypothesis, we have $(Q)\ \sigma \sqsubseteq \alpha$ $\mathbf{(\underline{1})}$, or $(Q)\ \alpha \sqsubseteq \sigma$ $\mathbf{(\underline{2})}$, and $\alpha \in \mathsf{ftv}(\sigma)$, that is, there exists u such that $\sigma/u = \alpha$. If u is ϵ, then we get the expected result by Property 2.1.5.ii. Otherwise, we have $u \neq \epsilon$ $\mathbf{(\underline{3})}$ and $(\forall\,(Q)\ \sigma) \cdot u/ = (\forall\,(Q)\ \alpha) \cdot \epsilon/$ $\mathbf{(\underline{4})}$ by definition of occurrences. If (1) holds, we get $\forall\,(Q)\ \sigma/ \leqslant_{/} \forall\,(Q)\ \alpha/$ by Property 2.1.3.ii. Hence, we get $(\forall\,(Q)\ \sigma) \cdot \epsilon/ \leqslant_{/} (\forall\,(Q)\ \sigma) \cdot u/$ by (4), and we conclude by Property i that u is ϵ. Otherwise, (2) holds, and we get $(Q)\ \alpha \equiv \sigma$ by Lemma 2.1.6. Hence, $\forall\,(Q)\ \alpha/ = \forall\,(Q)\ \sigma/$ $\mathbf{(\underline{5})}$ holds by Property 1.5.4.i (page 50). By (5) and (4), we get $(\forall\,(Q)\ \sigma) \cdot u/ = (\forall\,(Q)\ \sigma) \cdot \epsilon/$. This leads to $u = \epsilon$ by Property i. In both cases, we have $u = \epsilon$, which is a contradiction with (3). ■

2.2 Canonical representatives and bounds in a prefix

In the prefix $(\alpha \geq \bot, \beta = \alpha)$, we can see β as an alias for α. The following definition captures the intuition that α and β have the same representative:

Definition 2.2.1 Given a prefix Q and a type variable α in $\mathsf{dom}(Q)$, we define the representative of α in Q, which we write $Q[\alpha]$, by the following induction:

$$(Q, \alpha \diamond \sigma, Q')[\alpha] \triangleq \begin{cases} Q[\beta] & \text{if } \mathsf{nf}(\sigma) = \beta \\ \alpha & \text{otherwise} \end{cases}$$

The bound of $Q[\alpha]$ in Q is written $Q(\alpha)$. Note that if $Q(\alpha)$ is defined, it is not in $\mathcal{V}$.□

Intuitively, $Q(\alpha)$ is the "real" bound of α in Q, and $Q[\alpha]$ is a type variable equivalent to α under Q whose bound is the "real" bound of α. As an example, taking $Q \triangleq (\alpha \geq \bot, \beta = \alpha)$, we have $Q[\alpha] = \alpha$, $Q[\beta] = \alpha$, $Q(\alpha) = Q(\beta) = \bot$.

Properties 2.2.2 *We have the following properties for any closed well-formed prefix Q, and α, β in $\mathsf{dom}(Q)$:*

i) $(Q)\ \alpha \equiv Q[\alpha]$.
ii) *If $\widehat{Q}(\alpha) = \beta$, then we have $Q[\alpha] = \beta$ and $Q(\alpha) = Q(\beta)$ follows.*
iii) *If $\widehat{Q}(\alpha) \notin \mathcal{V}$, then $(Q)\ \alpha \equiv Q(\alpha)$ holds.*
iv) *We have $Q(\alpha) \notin \mathcal{T}$ iff $\widehat{Q}(\alpha) \in \vartheta$*
v) *If $(Q)\ \alpha \equiv \beta$, then $(Q)\ Q(\alpha) \equiv Q(\beta)$.*
vi) *If $(Q)\ \alpha \diamondsuit \sigma$ and $Q(\alpha) \notin \mathcal{T}$ hold, then $\mathsf{nf}(\sigma) \in \vartheta$ and $Q(\mathsf{nf}(\sigma)) \notin \mathcal{T}$.*

See proof in the Appendix (page 235).

2.3 Restricted and thrifty derivations

We wish to put equivalence, abstraction, and instance derivations in canonical forms. In this section, we introduce some syntactic restrictions on the derivations. We prove that these restrictions do not actually reduce the expressiveness of relations.

Restricted derivations Some rules can be restricted by adding some side-conditions that do not reduce the set of derivable judgments. Reasoning on restricted derivations often simplifies proofs.

Lemma 2.3.1 *Given a derivation of $(Q)\ \sigma \diamondsuit \sigma'$, there exists a derivation of the same judgment such that the following hold:*

- *In Rule* I-Bot, $\mathsf{nf}(\sigma)$ *is not* $\bot$ *and* $\sigma \notin \mathcal{V}$.
- *In Rules* R-Context-L, R-Context-Rigid, R-Context-Flexible, α *is in* $\mathsf{ftv}(\sigma)$ *and* $\mathsf{nf}(\sigma)$ *is not* α.
- *In Rule* I-Rigid, $\alpha \in \mathsf{ftv}(\sigma)$, σ_1 *is not in* $\mathcal{T}$, *and* $\mathsf{nf}(\sigma)$ *is not* α.
- *In rules* I-Hyp *and* A-Hyp, σ_1 *is not in* $\mathcal{T}$.

A derivation which follows those restrictions is called a restricted *derivation.*

The proof is by induction on the initial derivation. It should be simply remarked that whenever a rule is not restricted, as described above, it can be freely replaced by an equivalence. See proof in the Appendix (page 236).

Thrifty derivations Consider the following occurrence of the right-context rule. The symbol $\diamondsuit$ stands for either $\equiv$, $\sqsubseteq$, or $\sqsubseteq$:

$$\text{R-Context-R}\quad \frac{(Q, \alpha \diamond \sigma)\ \sigma_1 \diamondsuit \sigma_2}{(Q)\ \forall (\alpha \diamond \sigma)\ \sigma_1 \diamondsuit \forall (\alpha \diamond \sigma)\ \sigma_2}$$

This occurrence is said to be *squandering* when either $\mathsf{nf}(\sigma_1)$ or $\mathsf{nf}(\sigma_2)$ is α, but not both simultaneously. We will show in Lemma 2.3.3 that squandering occurrences of R-Context-R can always be removed.

Definition 2.3.2 (Thrifty Derivations) A derivation with no squandering rules is a *thrifty* derivation. □

This means that, in a thrifty derivation each occurrence of Rule R-Context-R (such as the one given above) $\mathsf{nf}(\sigma_1)$ is α if and only if $\mathsf{nf}(\sigma_2)$ is α.

Lemma 2.3.3 *Any derivation can be rewritten into a thrifty restricted derivation.*

The proof proceeds by rewriting derivations into thrifty derivations, and then by observing that restricted derivations are kept restricted. See the details of the proof in the Appendix (page 238).

The following corollary characterizes abstraction. It states that if a type σ_1 that is not a variable can be instantiated to a variable α, then it means that σ_1 has been abstracted by α. This must be done in two steps: first, σ_1 is instantiated to the bound of α, then it is abstracted by A-Hyp or I-Hyp. As a consequence, we know that the bound of α is an instance of σ_1.

Corollary 2.3.4 *Assume* $(\alpha \diamond \sigma) \in Q$ *and* $\sigma_1 \notin \mathcal{V}$*. If* $(Q)\ \sigma_1 \Diamond \alpha$ *holds, then so does* $(Q)\ \sigma_1 \Diamond \sigma$*. Besides, if* $\Diamond$ *is* $\sqsubseteq$*, then* $\diamond$ *is rigid.*

See proof in the Appendix (page 242).

2.4 Contexts

An occurrence points to a location in the skeleton of a type: the projection function cannot provide more information than skeletons do. However ML$^{\mathrm{F}}$ types are richer than skeletons, thus occurrences are not accurate enough. In order to point to a location in a type more accurately, we define *contexts*, which have the same structure as types.

We first define *generic contexts*, which are the most general form of contexts with a single hole. We also define special forms of generic contexts that are *narrow contexts*, *flexible contexts* and *rigid contexts*. Finally, we define contexts with multiple holes.

Generic contexts *Generic one-hole contexts* are defined by the following grammar:

$$C ::= [\,] \mid \forall\,(\alpha \diamond \sigma)\ C \mid \forall\,(\alpha \diamond C)\ \sigma$$

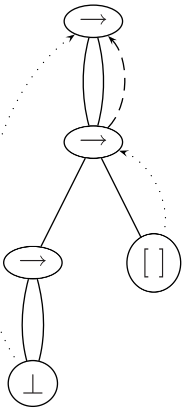
A well-formed context does not bind the same variable twice. The prefix of a well-formed context C, written $\overline{C}$, is defined recursively by

$$\overline{[\,]} \triangleq \emptyset \qquad \overline{\forall\,(\alpha \diamond \sigma)\ C} \triangleq (\alpha \diamond \sigma)\overline{C} \qquad \overline{\forall\,(\alpha \diamond C)\ \sigma} \triangleq \overline{C}$$

We write $\mathsf{dom}(C)$ for $\mathsf{dom}(\overline{C})$, and $\widehat{C}$ for $\widehat{\overline{C}}$. If C is well-formed, then $\overline{C}$ is well-formed. For example, let C be the context

$$\forall\,(\alpha \geq \bot, \beta = \forall\,(\gamma = \alpha \to \alpha)\ \forall\,(\delta \geq [\,])\ \gamma \to \delta)\ \beta \to \beta$$

Then $\overline{C}$ is $(\alpha \geq \bot, \gamma = \alpha \to \alpha)$, and $\widehat{C}$ is $[\alpha \to \alpha/\gamma]$. We can informally represent this context by the graph below. The node labelled $\bot$ corresponds to α, the node labelled $[\,]$ corresponds to δ. The former is flexible and bound at top-level; and the latter is flexible and bound at β. The middle node labelled $\to$ represents β; it is rigid and bound at top-level. The bottom node labelled $\to$ represents γ. It is a monotype, thus it is not bound.



The next definition introduces the notion of a useful context and of equivalence between contexts.

Definition 2.4.1

- A context C is said *useless* if, for all σ_1 and σ_2, $C(\sigma_1) \equiv C(\sigma_2)$ holds. Otherwise, it is said *useful*.
- Two contexts C_1 and C_2 are said $\bar{\alpha}$*-equivalent* if and only if $C_1(\sigma) \equiv C_2(\sigma)$ holds for any σ such that $\mathsf{ftv}(\sigma) \cap (\mathsf{dom}(C_1) \cup \mathsf{dom}(C_2)) \subseteq \bar{\alpha}$.
- Two contexts C_1 and C_2 are said *equivalent* if and only if $C_1(\sigma) \equiv C_2(\sigma)$ holds for any σ. □

For example, $\forall\,(\alpha{=}[\,])\ \beta \to \beta$ is useless, whereas $\forall\,(\alpha{=}[\,])\ \alpha \to \beta$ is useful. Additionally, the contexts $\forall\,(\alpha \geq \bot, \beta \geq \bot)\ [\,]$ and $\forall\,(\alpha \geq \bot, \gamma \geq \bot)\ [\,]$ are α-equivalent, but are not β- or γ-equivalent. Note that C_1 equivalent to C_2 means that C_1 is ϑ-equivalent to C_2.

We define next the level of a context C. It is intuitively the depth of the hole in C. More precisely, it will be shown that a useless context has level 0 (and conversely), and a context of the form $\forall\,(Q)\ [\,]$ has level 1 (and conversely). For example, a context such as $\forall\,(\alpha \geq [\,])\ \alpha \to \alpha$ has level 2, and a context such as $\forall\,(\alpha \geq \forall\,(\beta \geq [\,])\ \beta \to \beta)\ \alpha \to \alpha$ has level 3.

Then we define the level-1 domain of a context C. It corresponds intuitively to the type variables bound directly in front of the hole. For example, the level-1 domain of $\forall\,(\alpha \geq \bot, \beta \geq \forall\,(\gamma = \alpha \to \alpha, \delta \geq \bot)\ [\,])\ \alpha \to \beta$ is the set $\{\gamma, \delta\}$. Indeed, the bindings $(\gamma = \alpha \to \alpha, \delta \geq \bot)$ are in front of the hole $[\,]$. They are, intuitively, at the same level as the hole. Other bindings, such as $(\alpha \geq \bot)$ are not at the same level, thus α do not appear in the level-1 domain.

Definition 2.4.2

- The *level* of a context C, written $\mathsf{level}(C)$, is defined inductively as follows:

$$\mathsf{level}([\,]) \triangleq 1 \qquad \mathsf{level}(\forall\,(\alpha \diamond C)\ \sigma) \triangleq \begin{cases} \mathsf{level}(C) & \text{if } \sigma \equiv \alpha \\ 0 & \text{if } \alpha \notin \mathsf{ftv}(\sigma) \\ 0 & \text{if } \mathsf{level}(C) = 0 \\ \mathsf{level}(C) + 1 & \text{otherwise} \end{cases}$$

$$\mathsf{level}(\forall\,(\alpha \diamond \sigma)\ C) \triangleq \mathsf{level}(C)$$

- The level-*1 domain* of a context C, written $\mathsf{dom}_1(C)$, is defined inductively as follows:

$$\mathsf{dom}_1(C) \triangleq \emptyset \text{ if } \mathsf{level}(C) = 0 \qquad \mathsf{dom}_1(C) \triangleq \mathsf{dom}(C) \text{ if } \mathsf{level}(C) = 1$$

$$\mathsf{dom}_1(C) \triangleq \mathsf{dom}_1(C') \text{ if } \mathsf{level}(C) > 1 \text{ and } \begin{cases} C = \forall\,(\alpha \diamond \sigma)\ C' \\ \text{or} \\ C = \forall\,(\alpha \diamond C')\ \sigma \end{cases}$$

□

Properties 2.4.3

i) *If C is a useful context, then there exists u such that for any σ and u' in $\mathsf{dom}(\sigma)$, we have $C(\sigma)/uu' = \sigma/u'$.*
ii) *If C is a useful context, and $\sigma \notin \mathcal{T}$, then $C(\sigma) \notin \mathcal{T}$.*
iii) *A context C is useful iff $\mathsf{level}(C) > 0$.*

See proof in the Appendix (page 242).

We introduce three restrictions of generic contexts, namely narrow, flexible, and rigid contexts. Narrow contexts are intuitively contexts of the form $\forall\,(Q)\ [\,]$. However, thanks to rule Eq-Var, a context C_n such as $\forall\,(\alpha \geq \bot, \beta \geq \forall\,(Q')\ [\,])\ \beta$ is equivalent to $\forall\,(\alpha \geq \bot, Q')\ [\,]$. Hence, a context such as C_n is also considered narrow. This is captured by the following definition.

Narrow contexts *Narrow contexts* are a restriction of generic contexts defined by the following grammar:

$$C_n ::= [\,] \mid \forall\,(\alpha \diamond \sigma)\ C_n \mid \forall\,(\alpha \diamond C_n)\ \sigma \text{ where } \mathsf{nf}(\sigma) = \alpha$$

The following properties explain what a narrow context is.

Properties 2.4.4

- ***i)*** *A prefix C is narrow iff* $\mathsf{level}(C) = 1$.
- ***ii)*** *If* $\mathsf{level}(C) = 1$*, then C is equivalent to* $\forall\,(\overline{C})\ [\,]$.
- ***iii)*** *If* $\mathsf{level}(C) > 1$*, then there exists C' such that C is equivalent to* $C'(\forall\,(\alpha \diamond \forall\,(Q_1)\ [\,])\ \sigma)$ *and* $\mathsf{level}(C') = \mathsf{level}(C) - 1$.
- ***iv)*** *If* $\mathsf{level}(C) \geqslant 1$*, then there exists C' and C_n such that C is equivalent to $C'(C_n)$, and* $\mathsf{level}(C_n) = 1$, $\mathsf{dom}_1(C') = \emptyset$*, and* $\mathsf{dom}_1(C) = \mathsf{dom}(C_n)$.
- ***v)*** *If $C(\sigma) \in \mathcal{V}$, then* $\mathsf{level}(C) \leqslant 1$.

These properties are shown by structural induction on the context C.
See details in Appendix (page 243).

Intuitively, instantiation occurs only at flexible occurrences, that is, only in flexible bounds. It can also occur in the bound of α in $\forall\,(\alpha = \sigma)\ \alpha$, because it is equivalent to σ by Eq-Var. Flexible contexts, defined next, are contexts suitable for instantiation.

Flexible contexts *Flexible contexts* are a restriction of generic contexts defined by the following grammar:

$$C_f ::= [\,] \mid \forall\,(\alpha \diamond \sigma)\ C_f \mid \forall\,(\alpha \geq C_f)\ \sigma \mid \forall\,(\alpha = C_f)\ \sigma \text{ where } \mathsf{nf}(\sigma) = \alpha$$

Similarly, abstraction occurs only at rigid occurrences, that is, only in rigid bounds. Rigid contexts are contexts suitable for abstraction.

Rigid contexts *Rigid contexts* are a restriction of generic contexts defined by the following grammar:

$$C_r ::= [\,] \mid \forall\,(\alpha \diamond \sigma)\ C_r \mid \forall\,(\alpha = C_r)\ \sigma \mid \forall\,(\alpha \geq C_r)\ \sigma \text{ where } \mathsf{nf}(\sigma) = \alpha$$

The following lemma shows that the normal form of a type $C(\sigma_1)$ is of the form $C'(\mathsf{nf}(\widehat{C}(\sigma_1)))$ for some context C'. Intuitively, C' is some sort of normal form of C, which is partially captured by the statement that C' is $\bar{\alpha}$-equivalent to C (where the interface $\bar{\alpha}$ is the set of free variables of $\widehat{C}(\sigma_1)$). For instance, take the following:

$$C \triangleq \forall\,(\alpha \geq \bot, \beta = \alpha \to \alpha, \gamma \geq [\,])\ \beta \to \gamma \qquad\qquad \sigma_1 \triangleq \forall\,(\delta \geq \bot)\ \delta \to \beta$$

$$\sigma \triangleq C(\sigma_1) = \forall\,(\alpha \geq \bot, \beta = \alpha \to \alpha, \gamma \geq \forall\,(\delta \geq \bot)\ \delta \to \beta)\ \beta \to \gamma$$

We have $\widehat{C} = [\alpha \to \alpha/\beta]$. Then the normal form of σ is $\forall\,(\alpha \geq \bot, \gamma \geq \forall\,(\delta \geq \bot)\ \delta \to (\alpha \to \alpha))\ (\alpha \to \alpha) \to \gamma$. Taking $C' = \forall\,(\alpha \geq \bot, \gamma \geq [\,])\ (\alpha \to \alpha) \to \gamma$, we have $\mathsf{nf}(\sigma) = C'(\forall\,(\delta \geq \bot)\ \delta \to (\alpha \to \alpha))$, that is, $\mathsf{nf}(\sigma) = C'(\widehat{C}(\sigma_1))$. Additionally, C' and C are α-equivalent.

Lemma 2.4.5 *If C is a useful well-formed context, $\sigma = C(\sigma_1)$ and $\sigma_1 \notin \mathcal{T}$, then there exists C' such that*

- *C' is $\mathsf{ftv}(\widehat{C}(\sigma_1))$-equivalent to C*
- $\mathsf{dom}(C') \subseteq \mathsf{dom}(C)$
- $\mathsf{nf}(\sigma) = C'(\mathsf{nf}(\widehat{C}(\sigma_1)))$

Proof: This is proved by induction on the number of quantifiers of the context C. By hypothesis, $\sigma_1 \notin \mathcal{T}$ **(1)**. Let $\bar\alpha$ be $\mathsf{ftv}(\widehat{C}(\sigma_1))$ **(2)**. We proceed by case analysis on the structure of C.

○ Case C is $[\ ]$: Then σ is σ_1 and taking $C' = [\ ]$ is appropriate.

○ Case C is $\forall\,(\alpha_0 \diamond \sigma_0)\ C_0$ **(3)**: Then σ is $\forall\,(\alpha_0 \diamond \sigma_0)\ C_0(\sigma_1)$ **(4)**. Necessarily, C_0 is useful. By well-formedness of C, we must have $\alpha_0 \notin \mathsf{dom}(C_0)$ **(5)**. We have $\alpha_0 \notin \bar\alpha$ by (2) and well-formedness of (3).

Subcase $C_0(\sigma_1) \equiv \alpha_0$: By Property 1.5.11.ii (page 54), we have $C_0(\sigma_1) \in \mathcal{T}$. By Property 2.4.3.ii and (1), we have $C_0(\sigma_1) \notin \mathcal{T}$. This is a contradiction, which means that this subcase cannot occur.

Subcase $\sigma_0 \in \mathcal{T}$: By Property 1.5.11.ii (page 54), we have $\sigma_0 \equiv \tau_0$ for some monotype τ_0. Let θ be $[\tau_0/\alpha_0]$. The normal form of σ is by definition $\theta(\mathsf{nf}(C_0(\sigma_1)))$, from (4). By Property 1.5.6.iii (page 51), we get $\mathsf{nf}(\sigma) = \mathsf{nf}(\theta(C_0(\sigma_1)))$, that is, $\mathsf{nf}(\sigma) = \mathsf{nf}(\theta(C_0)(\theta(\sigma_1)))$ **(6)** thanks to (5). Observing that the substitution $\widehat{C}$ is $\theta \circ \widehat{C_0}$ **(7)**, we get $\mathsf{ftv}(\widehat{\theta(C_0)}(\theta(\sigma_1))) = \mathsf{ftv}(\widehat{C}(\sigma_1)) = \bar\alpha$. By induction hypothesis on $\theta(C_0)$, there exists a context C_1 that is $\bar\alpha$-equivalent to $\theta(C_0)$ **(8)** and such that $\mathsf{nf}(\theta(C_0)(\theta(\sigma_1))) = C_1(\mathsf{nf}(\widehat{\theta(C_0)}(\theta(\sigma_1))))$ **(9)** and $\mathsf{dom}(C_1) \subseteq \mathsf{dom}(\theta(C_0))$, that is, $\mathsf{dom}(C_1) \subseteq \mathsf{dom}(C_0)$ **(10)**. By (9), (6), and (7), $\mathsf{nf}(\sigma)$ is $C_1(\mathsf{nf}(\widehat{C}(\sigma_1)))$. Hence, $\theta(C_0)$ is $\bar\alpha$-equivalent to $\forall\,(\alpha_0 \diamond \sigma_0)\ C_0$, that is, to C **(11)**. By (8) and (11), C_1 is $\bar\alpha$-equivalent to C. Finally, $\mathsf{dom}(C_1) \subseteq \mathsf{dom}(C)$ by (10) and (3). This is the expected result.

Subcase $\alpha_0 \notin \mathsf{ftv}(C_0(\sigma_1))$ **(12)**: Then $\mathsf{nf}(\sigma)$ is $\mathsf{nf}(C_0(\sigma_1))$, thus we conclude directly by induction hypothesis. Indeed, C is $\forall\,(\alpha_0 \diamond \sigma_0)\ C_0$ and is $\bar\alpha$-equivalent to C_0 by (12). Besides, $\mathsf{dom}(C_0) \subseteq \mathsf{dom}(C)$, and $\widehat{C_0} = \widehat{C}$.

Otherwise $\mathsf{nf}(\sigma)$ is $\forall\,(\alpha_0 \diamond \mathsf{nf}(\sigma_0))\ \mathsf{nf}(C_0(\sigma_1))$, and $\widehat{C} = \widehat{C_0}$ **(13)**. By induction hypothesis, there exists C_1, $\bar\alpha$-equivalent to C_0, such that $\mathsf{nf}(C_0(\sigma_1)) = C_1(\mathsf{nf}(\widehat{C_0}(\sigma_1)))$. By (13), we get $\mathsf{nf}(C_0(\sigma_1)) = C_1(\mathsf{nf}(\widehat{C}(\sigma_1)))$. We get the expected result by taking $C' = \forall\,(\alpha_0 \diamond \mathsf{nf}(\sigma_0))\ C_1$. Besides, C' is equivalent to $\forall\,(\alpha_0 \diamond \sigma_0)\ C_1$, which is $\bar\alpha$-equivalent to $\forall\,(\alpha_0 \diamond \sigma_0)\ C_0$, that is C. Finally, $\mathsf{dom}(C') = \{\alpha_0\} \cup \mathsf{dom}(C_1)$ and $\mathsf{dom}(C_1) \subseteq \mathsf{dom}(C_0)$ by induction hypothesis, thus $\mathsf{dom}(C') \subseteq \{\alpha_0\} \cup \mathsf{dom}(C_0) = \mathsf{dom}(C)$.

○ Case C is $\forall\,(\alpha_0 \diamond C_0)\ \sigma_0$: then, σ is $\forall\,(\alpha_0 = C_0(\sigma_1))\ \sigma_0$. We must have $\alpha_0 \in \mathsf{ftv}(\sigma_0)$ and C_0 is useful (otherwise C would be useless). Moreover, $\widehat{C} = \widehat{C_0}$ **(14)** by definition.

SUBCASE $\mathsf{nf}(\sigma_0)$ is α_0: then, $\mathsf{nf}(\sigma)$ is $\mathsf{nf}(C_0(\sigma_1))$ and we conclude directly by induction hypothesis. Indeed, C is equivalent to C_0, $\mathsf{dom}(C_0) = \mathsf{dom}(C)$, and $\widehat{C} = \widehat{C_0}$ from (14).

SUBCASE $C_0(\sigma_1) \in \mathcal{T}$ is not possible by Property 2.4.3.ii.

OTHERWISE $\mathsf{nf}(\sigma)$ is $\forall\,(\alpha_0 \diamond \mathsf{nf}(C_0(\sigma_1)))\ \mathsf{nf}(\sigma_0)$. By induction hypothesis, there exists C_1, $\bar{\alpha}$-equivalent to C_0 such that $\mathsf{nf}(C_0(\sigma_1)) = C_1(\mathsf{nf}(\widehat{C_0}(\sigma_1)))$ **(<u>15</u>)** and $\mathsf{dom}(C_1) \subseteq \mathsf{dom}(C_0)$ **(<u>16</u>)**. Hence, by (15) and (14), we have $\mathsf{nf}(C_0(\sigma_1)) = C_1(\mathsf{nf}(\widehat{C}(\sigma_1)))$. We get the expected result by taking $C' = \forall\,(\alpha_0 \diamond C_1)\ \mathsf{nf}(\sigma_0)$. Besides, C' is equivalent to $\forall\,(\alpha_0 \diamond C_1)\ \sigma_0$, which is $\bar{\alpha}$-equivalent to $\forall\,(\alpha_0 \diamond C_0)\ \sigma_0$, that is, C. From (16), we get $\mathsf{dom}(C_1) \subseteq \mathsf{dom}(C)$. Finally, $\mathsf{dom}(C') = \mathsf{dom}(C_1) \subseteq \mathsf{dom}(C)$ holds. This is the expected result. ■

Contexts with multiple holes Generic contexts with n holes C^n are defined by the following grammar:

$$\begin{aligned} C^0 &::= \sigma && \text{no hole} \\ C^1 &::= C && \\ C^{m+p} &::= \forall\,(\alpha \diamond C^m)\ C^p && \text{for } m+p>1. \end{aligned}$$

The notation $C^2(\sigma_1, \sigma_2)$ means that the first hole is filled with σ_1 and the second hole filled with σ_2.[1] Note that the notion of useful context is more delicate: the "usefulness" of the first hole might depend on what is put in the second hole. For instance, take $\forall\,(\alpha = [\,])\ [\,]$.

Matching contexts We say that two generic contexts C_1 and C_2 match on σ if there exist σ_1 and σ_2 such that $\sigma = C_1(\sigma_1) = C_2(\sigma_2)$. Two matching contexts C_1 and C_2 are *nested* when there exists C_3 such that $C_1 = C_2(C_3)$ or $C_2 = C_1(C_3)$. Otherwise, they are said disjoint.

Lemma 2.4.6 *Assume σ is in normal form. If C_1 and C_2 are disjoint and match on σ, then there exist a two-hole context C^2 such that $C_1 = C^2([\,], \sigma_2)$ and $C_2 = C^2(\sigma_1, [\,])$. Besides, if C_1 is flexible (resp. rigid), then $C^2([\,], \sigma_2')$ is flexible (resp. rigid), for any σ_2'. A similar result holds for C_2.*

<u>Proof</u>: By induction on the structures of C_1 and C_2. By hypothesis, we have $\sigma = C_1(\sigma_1) = C_2(\sigma_2)$ **(<u>1</u>)**, and C_1 and C_2 are not nested.

---

[1] We should explain formally how to assign a number to each hole. However, this would only obfuscate the notations. Hence, we leave some harmless "artistic blur", and assume that we have a way to assign a different number to each hole in a context.

- Case $C_1 = [\,]$ is not possible, since otherwise C_1 and C_2 are nested.

- Case $C_2 = [\,]$ is not possible either.

- Case $C_1 = \forall\,(\alpha \diamond \sigma)\; C_1'$, $C_2 = \forall\,(\alpha' \diamond' \sigma')\; C_2'$: Then (1) implies $(\alpha \diamond \sigma) = (\alpha' \diamond' \sigma')$ and $C_1'(\sigma_1) = C_2'(\sigma_2)$. By induction hypothesis, there exists C_a^2 such that $C_1' = C_a^2([\,], \sigma_2)$ and $C_2' = C_a^2(\sigma_1, [\,])$. Hence taking $C_2 = \forall\,(\alpha \diamond \sigma)\; C_a^2([\,],[\,])$ gives the expected result. Besides, if C_1 is flexible (resp. rigid), then C_1' is flexible (resp. rigid). By induction hypothesis, $C_a^2([\,], \sigma_2')$ is flexible (resp. rigid) for any σ_2'. Hence, $C_2([\,], \sigma_2')$ is flexible (resp. rigid). Similarly for $C_2(\sigma_1', [\,])$.

- Case $C_1 = \forall\,(\alpha \diamond \sigma)\; C_1'$, $C_2 = \forall\,(\alpha' \diamond' C_2')\; \sigma'$: Then (1) implies $(\alpha \diamond \sigma) = (\alpha' \diamond' C_2'(\sigma_2))$ and $\sigma' = C_1'(\sigma_1)$. Let C^2 be $\forall\,(\alpha \diamond C_2')\; C_1'$, where the first hole is in C_1', the second hole in C_2'. We have $C_1 = C^2([\,], \sigma_2)$ and $C_2 = C^2(\sigma_1, [\,])$, which is the expected result. Besides, if C_1 is flexible (resp. rigid), then C_1' is flexible (resp. rigid). Hence, $C_2([\,], \sigma_2')$ is flexible (resp. rigid). If C_2 is flexible (resp. rigid), then C_2' is flexible (resp. rigid) and $\diamond'$ is $\geq$ (resp $=$). The case where $\mathtt{nf}(\sigma')$ is α' and $\diamond'$ is $=$ (resp $\geq$) is not possible here, because σ is in normal form by hypothesis. Hence, $C^2(\sigma_1', [\,])$ is flexible (resp. rigid).

- Case $C_1 = \forall\,(\alpha \diamond C_1')\; \sigma$, $C_2 = \forall\,(\alpha' \diamond' \sigma')\; C_2'$ is similar.

- Case $C_1 = \forall\,(\alpha \diamond C_1')\; \sigma$, $C_2 = \forall\,(\alpha' \diamond' C_2')\; \sigma'$: Then (1) implies $\alpha = \alpha'$, $\diamond = \diamond'$, $\sigma = \sigma'$, and $C_1'(\sigma_1) = C_2'(\sigma_2)$. Thus, by induction hypothesis there exists C_a^2 such that $C_1' = C_a^2([\,], \sigma_2)$ and $C_2' = C_a^2(\sigma_1, [\,])$. Hence, taking $C_2 = \forall\,(\alpha \diamond C_a^2)\; \sigma$ gives the expected result. Besides, if C_1 is flexible (resp. rigid), then $\diamond$ is $\geq$ (resp. $=$) and C_1' is flexible (resp. rigid). By induction hypothesis, $C_a^2([\,], \sigma_2')$ is flexible (resp. rigid), for any σ_2'. Hence, $C_2([\,], \sigma_2')$ is flexible (resp. rigid) for any σ_2'. Similarly for $C_2(\sigma_1', [\,])$. ■

Derivable context rules The following rules are derivable by induction on the context. They can represent any sequence of context rules.

Eq-Context*

$$\frac{(Q\overline{C})\ \sigma_1 \equiv \sigma_2}{(Q)\ C(\sigma_1) \equiv C(\sigma_2)}$$

A-Context*

$$\frac{(Q\overline{C_r})\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma_2}{(Q)\ C_r(\sigma_1) \mathrel{\sqsubseteq\!\!\!\!\!-} C_r(\sigma_2)}$$

I-Context*

$$\frac{(Q\overline{C_f})\ \sigma_1 \sqsubseteq \sigma_2}{(Q)\ C_f(\sigma_1) \sqsubseteq C_f(\sigma_2)}$$

2.5 Local abstraction and instance rules

The rules defining abstraction and instance were given in Figure 1.2 and 1.3, in Sections 1.6 and 1.7. In order to show confluence results on abstraction and instantiation, we need to put derivations in canonical forms. This amounts to lifting transitivity to top-level, and to replacing some rules, such as A-Hyp by local rules. Besides, we wish to get rid of context rules such as R-Context-R and R-Context-Rigid by using *contexts* defined in the previous section.

We recall that $\mathcal{R}^*$ is the transitive closure of $\mathcal{R}$.

2.5.1 Context-based rules

The following lemma states that transitivity can always be lifted to top-level.

Lemma 2.5.1 *If we have a derivation of* $(Q)\ \sigma \diamondsuit \sigma'$*, then, there exists a tuple* $\sigma_1 \ldots \sigma_n$ *such that* $\sigma_1 = \sigma$*,* $\sigma_n = \sigma'$ *and for all* $i < n$ *we have a derivation of* $(Q)\ \sigma_i \diamondsuit \sigma_{i+1}$ *which does not use Rule* R-TRANS.

Proof: By induction on the derivation of $(Q)\ \sigma \diamondsuit \sigma'$. All cases are easy. ■

Abstraction

We wish to define a relation that has the same expressiveness as $\mathrel{-\mkern-12mu\sqsubseteq}$ and that uses *contexts* instead of context rules. We consider the relation $\mathrel{-\mkern-12mu\sqsubseteq}_C$ defined by the single following rule:

$$\frac{(\alpha = \sigma_1) \in Q\overline{C_r}}{(Q)\ C_r(\sigma_1) \mathrel{-\mkern-12mu\sqsubseteq}_C C_r(\alpha)}\ \textsc{Ac-Hyp}$$

Intuitively, $\mathrel{-\mkern-12mu\sqsubseteq}_C$ captures abstraction, but not equivalence. Hence, we define $(\equiv|\mathrel{-\mkern-12mu\sqsubseteq}_C)$ as $(\equiv) \cup (\mathrel{-\mkern-12mu\sqsubseteq}_C)$, that is:

$(Q)\ \sigma_1\ (\equiv|\mathrel{-\mkern-12mu\sqsubseteq}_C)\ \sigma_2$ *if and only if* $(Q)\ \sigma_1 \equiv \sigma_2$ *or* $(Q)\ \sigma_1 \mathrel{-\mkern-12mu\sqsubseteq}_C \sigma_2$ *holds.*

As expected, the transitive closure of $(\equiv|\mathrel{-\mkern-12mu\sqsubseteq}_C)$ is equivalent to abstraction.

Lemma 2.5.2 *The relations* $\mathrel{-\mkern-12mu\sqsubseteq}$ *and* $(\equiv|\mathrel{-\mkern-12mu\sqsubseteq}_C)^*$ *are equal.*

Proof: We need to show that $(\equiv|\mathrel{-\mkern-12mu\sqsubseteq}_C)^*$ is included in $\mathrel{-\mkern-12mu\sqsubseteq}$, and conversely.

Directly: Rule AC-HYP is derivable with A-HYP and A-CONTEXT*. Hence, $\mathrel{-\mkern-12mu\sqsubseteq}_C$ is included in $\mathrel{-\mkern-12mu\sqsubseteq}$. Moreover, $\equiv$ is included in $\mathrel{-\mkern-12mu\sqsubseteq}$ and $\mathrel{-\mkern-12mu\sqsubseteq}$ is transitive, thus $(\equiv|\mathrel{-\mkern-12mu\sqsubseteq}_C)^*$ is included in $\mathrel{-\mkern-12mu\sqsubseteq}$.

Conversely, we show that $\mathrel{-\mkern-12mu\sqsubseteq}$ is included in $(\equiv|\mathrel{-\mkern-12mu\sqsubseteq}_C)^*$. Thanks to Lemma 2.5.1, it suffices to show that if we have a derivation of $(Q)\ \sigma_1 \mathrel{-\mkern-12mu\sqsubseteq} \sigma_2$ not using R-TRANS, then $(Q)\ \sigma_1 \equiv \sigma_2$ or $(Q)\ \sigma_1 \mathrel{-\mkern-12mu\sqsubseteq}_C \sigma_2$ is derivable. We assume we have a derivation of $(Q)\ \sigma_1 \mathrel{-\mkern-12mu\sqsubseteq} \sigma_2$ that does not use R-TRANS. All abstractions rules of this derivation have at most one premise. Namely, the rules with one premise are context rules R-CONTEXT-R and R-CONTEXT-RIGID. The topmost abstraction rule is either A-EQUIV or A-HYP. If it is A-EQUIV, then $(Q)\ \sigma_1 \equiv \sigma_2$ is derivable, and this case is solved. Otherwise, we have a derivation of the form

$$\cfrac{\textsc{A-Hyp}\ \dfrac{(\alpha = \sigma_0) \in QQ'}{(QQ')\ \sigma_0 \mathrel{\sqsubseteq\!\!\!\!\!-} \alpha}}{\cfrac{\vdots}{(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma_2}}$$

By structural induction on this derivation, we can show that there exists a rigid context C_r such that $\sigma_1 = C_r(\sigma_0)$, $\sigma_2 = C_r(\alpha)$, and $Q' = \overline{C_r}$. Hence, we have $(\alpha{=}\sigma_0) \in Q\overline{C_r}$ and the derivation ends with $(Q)\ C_r(\sigma_0) \mathrel{\sqsubseteq\!\!\!\!\!-} C_r(\alpha)$. Consequently, we have $(Q)\ C_r(\sigma_0) \mathrel{\sqsubseteq\!\!\!\!\!-}_C C_r(\alpha)$ by Rule Ac-Hyp, that is, $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-}_C \sigma_2$. This is the expected result. ■

Instance

As we have done for $\mathrel{\sqsubseteq\!\!\!\!\!-}_c$, we define the relation $\sqsubseteq_C$ in order to capture type instantiation through context-based rules.

$$\textsc{Ic-Bot}\ \frac{}{(Q)\ C_f(\bot) \sqsubseteq_C C_f(\sigma)} \qquad \textsc{Ic-Hyp}\ \frac{(\alpha \geq \sigma) \in Q\overline{C_f}}{(Q)\ C_f(\sigma) \sqsubseteq_C C_f(\alpha)} \qquad \textsc{Ic-Abstract}\ \frac{(Q\overline{C_f})\ \sigma \mathrel{\sqsubseteq\!\!\!\!\!-}_C \sigma'}{(Q)\ C_f(\sigma) \sqsubseteq_C C_f(\sigma')}$$

$$\textsc{Ic-Rigid}\ \frac{}{(Q)\ C_f(\forall\,(\alpha \geq \sigma)\ \sigma' \sqsubseteq_C C_f(\forall\,(\alpha = \sigma)\ \sigma')}$$

The relation $(\equiv|\sqsubseteq_C)$ is defined as $(\equiv) \cup (\sqsubseteq_C)$.

Lemma 2.5.3 *The relations $\sqsubseteq$ and $(\equiv|\sqsubseteq_C)^*$ are equal.*

Proof: We need to show that $(\equiv|\sqsubseteq_C)^*$ is included in $\sqsubseteq$, and conversely.

Directly: rules Ic-Bot, Ic-Hyp, Ic-Rigid, and Ic-Abstract are derivable with I-Context* and I-Bot, I-Hyp, I-Rigid, and I-Abstract, respectively. Hence, $\sqsubseteq_C$ is included in $\sqsubseteq$. Moreover, $\equiv$ is included in $\sqsubseteq$ and $\sqsubseteq$ is transitive, thus $(\equiv|\sqsubseteq_C)^*$ is included in $\sqsubseteq$.

Conversely, we show that $\sqsubseteq$ is included in $(\equiv|\sqsubseteq_C)^*$. We assume that $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ **(1)** holds. We have to show that $(Q)\ \sigma_1\ (\equiv|\sqsubseteq_C)^*\ \sigma_2$ is derivable. Like in the proof of Lemma 2.5.2, we assume that the derivation of (1) does not use transitivity (thanks to Lemma 2.5.1). Hence, the derivation of (1) is actually a sequence of context-rules starting with a one-premise rule I-X, which can be either I-Abstract, I-Hyp, I-Bot or I-Rigid. Hence, the derivation of (1) is of the form:

$$\dfrac{\dfrac{\text{I-X}\ \dfrac{}{(QQ')\ \sigma'_1 \sqsubseteq \sigma'_2}}{\vdots}}{(Q)\ \sigma_1 \sqsubseteq \sigma_2}$$

In the first case (I-Abstract), following the proof of Lemma 2.5.2, we have a sequence of context rules (namely R-Context-Rigid or R-Context-R) starting with A-Hyp or A-Equiv. If it starts with A-Equiv, then $(Q)\ \sigma_1 \equiv \sigma_2$ holds, and the result is shown. Otherwise, and for other choices of I-X also, there exists a flexible context C_f such that $\overline{C_f} = Q'$, $\sigma_1 = C_f(\sigma'_1)$ and $\sigma_2 = C_f(\sigma'_2)$. Then we get the expected result by using rules (respectively) Ic-Abstract, Ic-Hyp, Ic-Bot and Ic-Rigid, and using Lemma 2.5.2 for I-Abstract. ■

2.5.2 Alternative abstraction and alternative instance

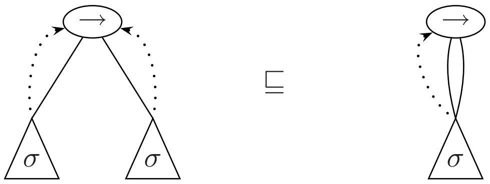
Rules A-Hyp and Ac-Hyp are not local. Indeed, both rules replace a type σ by a variable α, provided the binding $(\alpha{=}\sigma)$ is in the prefix. For example, $\forall\,(\alpha{=}\sigma)\ \forall\,(\beta{=}\sigma)\ \alpha \to \beta \sqsupseteq \forall\,(\alpha{=}\sigma)\ \forall\,(\beta{=}\alpha)\ \alpha \to \beta$ is derivable, taking $C_r = \forall\,(\alpha{=}\sigma)\ \forall\,(\beta{=}[\,])\ \alpha \to \beta$ in Ac-Hyp. In that case, the binding $(\alpha{=}\sigma)$ is next to the hole. However, the context C_r can be of any level, which means that the information $(\alpha = \sigma)$ is propagated from one subterm of the type to another arbitrarily distant subterm. We wish to force rigid contexts to be of level 1 or 2 only. This gives more granularity to derivations, and makes it possible to show some "local" invariants that would possibly be broken by non-local rules.

In order to help the understanding of the following, we introduce the informal notion of *frozen* binding: Given a derivation D ending with $(Q)\ \sigma_1 \diamondsuit \sigma_2$, we call Q the *frozen* prefix. Inside the derivation D we may find a sequent such as $(QQ')\ \sigma'_1 \diamondsuit \sigma'_2$. Whereas the bindings of Q are frozen, the bindings of Q', which were introduced by context rules, are not.

Pursuing the analysis, we see that Rule A-Hyp can be used with three different flavors:

1. Abstraction between types at the same level; *e.g.* the following is derivable (as seen above): $(Q)\ \forall\,(\alpha = \sigma)\ \forall\,(\beta = \sigma)\ \alpha \to \beta \sqsupseteq \forall\,(\alpha = \sigma)\ \forall\,(\beta = \alpha)\ \alpha \to \beta$.
2. Extrusion of a quantifier, as in A-Up*: $(Q)\ \forall\,(\alpha = \forall\,(\beta = \sigma)\ \sigma')\ \tau \sqsupseteq \forall\,(\beta = \sigma)\ \forall\,(\alpha = \sigma')\ \tau$, provided $\beta \notin \mathsf{ftv}(\tau)$.
3. Abstraction of a type found in the frozen prefix: $(\alpha = \sigma)\ \forall\,(\beta = \sigma)\ \beta \to \beta \sqsupseteq \forall\,(\beta = \alpha)\ \beta \to \beta$.

We have already illustrated the extrusion of a quantifier by graphs page 58. The abstraction between a type and the prefix can also be represented by the same graphs, considering the top-level node in the prefix. The abstraction between types at the same level can be captured by graphs as follows:



The same flavors, illustrated by similar examples, exist for I-Hyp. We now define two relations $\sqsubset\!\!\!\!-^{\bar\alpha}$ and $\sqsubseteq^{\bar\alpha}$ (frozen abstraction and frozen instance relations) that explicitly separate these different flavors. In order to keep track of frozen bindings, we annotate the relation with a set of variables $\bar\alpha$ that corresponds to variables introduced in the prefix by context rules (that is, non-frozen variables). Hence, we consider the following rule, which is meant to replace R-Context-R:

R-Context-R'

$$\frac{(Q, \alpha \diamond \sigma)\ \sigma_1 \diamondsuit^{\bar\alpha \cup \{\alpha\}}\ \sigma_2}{(Q)\ \forall\,(\alpha \diamond \sigma)\ \sigma_1 \diamondsuit^{\bar\alpha}\ \forall\,(\alpha \diamond \sigma)\ \sigma_2}$$

Definition 2.5.4 (Frozen abstraction and frozen instance) We define the relation $\sqsubset\!\!\!\!-^{\bar\alpha}$ as the smallest transitive relation under prefix that satisfies the rules of figure 2.1 as well as rules R-Context-Rigid and R-Context-R'.

The relation $\sqsubseteq^{\bar\alpha}$ is the smallest transitive relation under prefix that satisfies the rules of figure 2.2 as well as rules R-Context-Flexible and R-Context-R'. □

Note that $\sqsubset\!\!\!\!-^{\bar\alpha}$ is almost a rigid-compliant relation; the only difference lies in Rule R-Context-R', which is used instead of R-Context-R. Similarly, $\sqsubseteq^{\bar\alpha}$ is almost a flexible-compliant relation, the difference being in Rule R-Context-R'.

The rules for $\sqsubset\!\!\!\!-^{\bar\alpha}$ and $\sqsubseteq^{\bar\alpha}$ are similar to the rules defining $\sqsubset\!\!\!\!-$ and $\sqsubseteq$, except A-Hyp' and I-Hyp', which require the binding read in the prefix not to be in $\bar\alpha$ (that is, the binding must be frozen). In order to make local unification or extrusion of binders possible, rules A-Alias' and A-Up' are added to the definition of $\sqsubset\!\!\!\!-^{\bar\alpha}$, and rules I-Alias' and I-Up' are added to the definition of $\sqsubseteq^{\bar\alpha}$. They are meant to be used instead of A-Hyp or I-Hyp, when the binding read in the prefix is in $\bar\alpha$ (that is, not frozen). A judgment $(Q)\ \sigma_1 \diamondsuit^{\bar\alpha}\ \sigma_2$ is well formed if and only if Q is well-formed, and σ_1 and σ_2 are closed under Q. All derivations are implicitly well-formed. We note that $\sqsubset\!\!\!\!-^{\bar\alpha}$ is included in $\sqsubset\!\!\!\!-^{\bar\alpha\cup\bar\beta}$ for any sets of variables $\bar\alpha$ and $\bar\beta$. As expected, these new relations are equivalent to the original ones:

Figure 2.1: Frozen abstraction relation

A-Equiv'
$$\frac{(Q)\ \sigma_1 \equiv \sigma_2}{(Q)\ \sigma_1 \mathrel{\sqsubset\!\!\!\!\!=}^{\bar{\alpha}} \sigma_2}$$

A-Hyp'
$$\frac{(\alpha = \sigma) \in Q \qquad \alpha \notin \bar{\alpha}}{(Q)\ \sigma \mathrel{\sqsubset\!\!\!\!\!=}^{\bar{\alpha}} \alpha}$$

A-Up'
$$\frac{\alpha_1 \notin \mathsf{ftv}(\sigma')}{(Q)\ \forall\,(\alpha = \forall\,(\alpha_1 = \sigma_1)\ \sigma)\ \sigma' \mathrel{\sqsubset\!\!\!\!\!=}^{\bar{\alpha}} \forall\,(\alpha_1 = \sigma_1)\ \forall\,(\alpha = \sigma)\ \sigma'}$$

A-Alias'
$$\frac{}{(Q)\ \forall\,(\alpha_1 = \sigma_1)\ \forall\,(\alpha_2 = \sigma_1)\ \sigma \mathrel{\sqsubset\!\!\!\!\!=}^{\bar{\alpha}} \forall\,(\alpha_1 = \sigma_1)\ \forall\,(\alpha_2 = \alpha_1)\ \sigma}$$

Lemma 2.5.5 *The relations* $\mathrel{\sqsubset\!\!\!\!\!=}$ *and* $\mathrel{\sqsubset\!\!\!\!\!=}^{\emptyset}$ *are equal. The relations* $\sqsubseteq$ *and* $\sqsubseteq^{\emptyset}$ *are equal.*

See proof in the Appendix (page 244).

The following rule is derivable:

I-Equiv*'
$$\frac{(Q)\ \sigma_1 \equiv \sigma_2}{(Q)\ \sigma_1 \sqsubseteq^{\bar{\alpha}} \sigma_2}$$

Some proofs are made easier by considering restricted derivations for $\mathrel{\sqsubset\!\!\!\!\!=}^{\bar{\alpha}}$ and $\sqsubseteq^{\bar{\alpha}}$. The following lemma introduces new restrictions for the new rules A-Up, A-Alias, ...

Lemma 2.5.6 *The restrictions stated in Lemma 2.3.1 are still applicable with* $\sqsubseteq^{\bar{\alpha}}$ *and* $\mathrel{\sqsubset\!\!\!\!\!=}^{\bar{\alpha}}$*, together with a few more restrictions:*

- *In rules* A-Hyp' *and* I-Hyp', σ *is not in* $\mathcal{T}$.
- *In rules* A-Up' *and* I-Up', $\alpha \in \mathsf{ftv}(\sigma')$, $\mathsf{nf}(\sigma')$ *is not* α, $\alpha_1 \in \mathsf{ftv}(\sigma)$, $\mathsf{nf}(\sigma)$ *is not* α_1, *and* σ_1 *is not in* $\mathcal{T}$
- *In rules* A-Alias' *and* I-Alias', σ_1 *is not in* $\mathcal{T}$, *and* α_1 *and* α_2 *are in* $\mathsf{ftv}(\sigma)$

Proof: Similar to the proof of Lemma 2.3.1, and we only need to consider the new rules A-Hyp', I-Hyp', A-Up', I-Up', A-Alias', and I-Alias'. We assume given a judgment $(Q)\ \sigma_1 \Diamond^{\bar{\alpha}} \sigma_2$ derived with such a rule. If the restrictions above do not hold, then it is

Figure 2.2: Frozen instance relation

I-Abstract'
$$\frac{(Q)\ \sigma_1 \mathrel{-\!\!\!\!\sqsubseteq}^{\bar\alpha} \sigma_2}{(Q)\ \sigma_1 \sqsubseteq^{\bar\alpha} \sigma_2}$$

I-Bot'
$$(Q)\ \bot \sqsubseteq^{\bar\alpha} \sigma$$

I-Rigid'
$$\frac{}{(Q)\ \forall\,(\alpha \geq \sigma_1)\ \sigma \sqsubseteq^{\bar\alpha} \forall\,(\alpha = \sigma_1)\ \sigma}$$

I-Hyp'
$$\frac{(\alpha \geq \sigma) \in Q \qquad \alpha \notin \bar\alpha}{(Q)\ \sigma \sqsubseteq^{\bar\alpha} \alpha}$$

I-Up'
$$\frac{\alpha_1 \notin \mathsf{ftv}(\sigma')}{(Q)\ \forall\,(\alpha \geq \forall\,(\alpha_1 \diamond \sigma_1)\ \sigma)\ \sigma' \sqsubseteq^{\bar\alpha} \forall\,(\alpha_1 \diamond \sigma_1)\ \forall\,(\alpha \geq \sigma)\ \sigma'}$$

I-Alias'
$$\frac{}{(Q)\ \forall\,(\alpha_1 \geq \sigma_1)\ \forall\,(\alpha_2 \geq \sigma_1)\ \sigma \sqsubseteq^{\bar\alpha} \forall\,(\alpha_1 \geq \sigma_1)\ \forall\,(\alpha_2 = \alpha_1)\ \sigma}$$

easy to check that $(Q)\ \sigma_1 \equiv \sigma_2$ holds, and we conclude by Lemma 2.3.1, A-Equiv' or I-Equiv*'. ■

A derivation with $\sqsubseteq^{\bar\alpha}$ or $\mathrel{-\!\!\!\!\sqsubseteq}^{\bar\alpha}$ that keeps skeletons unchanged cannot instantiate $\bot$, but only replaces types by an alias (like A-Alias'), or extrudes binders (like A-Up'). Since A-Up' and A-Alias' do not modify the set of free variables, only A-Hyp can introduce new variables, which are necessarily frozen, that is, not in $\bar\alpha$. This is expressed by the following lemma:

Lemma 2.5.7 *If we have*

$$(Q)\ \sigma_1 \mathrel{\Diamond}^{\bar\alpha} \sigma_2 \qquad \forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/ \qquad \alpha \in \bar\alpha \qquad \alpha \in \mathsf{ftv}(\widehat{Q}(\sigma_2))$$

Then we have $\alpha \in \mathsf{ftv}(\widehat{Q}(\sigma_1))$.

The proof is by induction on the derivation of $(Q)\ \sigma_1 \mathrel{\Diamond}^{\bar\alpha} \sigma_2$.
See details in the Appendix (page 246).

2.6 Atomic instances

In this section, we define new relations, namely, $\dot\sqsubseteq$ and $\dot{\mathrel{-\!\!\!\!\sqsubseteq}}^{\bar\alpha}$, that have the same expressiveness as $\sqsubseteq$ and $\mathrel{-\!\!\!\!\sqsubseteq}$ but ensure canonical derivations. Besides, these relations are not transitive, so that all the steps of the instantiation are clearly separated. Furthermore, each step is always irreversible, in the sense that it is not an equivalence. In other words, all steps are useful.

Figure 2.3: Strict instance

In all rules, C_f is a well-formed useful context.

S-Hyp

$$\frac{\begin{array}{c}\sigma' \notin \mathcal{T} \\ (\alpha \geq \sigma_1) \in Q \qquad \alpha \notin \mathsf{dom}(C_f) \qquad (Q)\ \sigma_1 \equiv \widehat{C_f}(\sigma')\end{array}}{(Q)\ C_f(\sigma')\ \dot{\sqsubset}\ C_f(\alpha)}$$

S-Up

$$\frac{\begin{array}{c}\beta \in \mathsf{ftv}(\sigma'') \qquad \alpha \in \mathsf{ftv}(\sigma') \qquad \sigma', \sigma'' \notin \mathcal{V} \qquad \sigma \notin \mathcal{T} \\ \alpha \notin \mathsf{dom}(Q\overline{C_f}Q') \qquad \mathsf{ftv}(\sigma)\ \#\ \mathsf{dom}(Q')\end{array}}{(Q)\ C_f(\forall\,(\beta \geq \forall\,(Q', \alpha \diamond \sigma)\ \sigma')\ \sigma'')\ \dot{\sqsubset}\ C_f(\forall\,(\alpha \diamond \sigma)\ \forall\,(\beta \geq \forall\,(Q')\ \sigma')\ \sigma'')}$$

S-Alias

$$\frac{\begin{array}{c}\alpha_1 \in \mathsf{ftv}(\sigma') \qquad \alpha_2 \in \mathsf{ftv}(\sigma') \qquad \mathsf{dom}(Q')\ \#\ \mathsf{ftv}(\sigma_1, \sigma_2) \qquad \sigma_1 \notin \mathcal{T} \\ \sigma_1 \equiv \sigma_2 \qquad \sigma = \forall\,(\alpha_1 \geq \sigma_1, Q', \alpha_2 \geq \sigma_2)\ \sigma'\end{array}}{(Q)\ C_f(\sigma)\ \dot{\sqsubset}\ C_f(\forall\,(\alpha_1 \geq \sigma_1, Q', \alpha_2 = \alpha_1)\ \sigma')}$$

S-Nil

$$\frac{\mathsf{ftv}(\sigma) = \emptyset}{(Q)\ C_f(\bot)\ \dot{\sqsubset}\ C_f(\sigma)}$$

S-Rigid

$$\frac{\sigma' \notin \mathcal{T} \qquad \alpha \in \mathsf{ftv}(\sigma) \qquad \sigma \notin \mathcal{V}}{(Q)\ C_f(\forall\,(\alpha \geq \sigma')\ \sigma)\ \dot{\sqsubset}\ C_f(\forall\,(\alpha = \sigma')\ \sigma)}$$

2.6.1 Definitions

The one-step strict instance relation $\dot{\sqsubset}$ is defined in Figure 2.3. The one-step abstraction relation $\dot{\sqsubset\!\!\!\!-}^{\bar{\alpha}}$ is defined in Figure 2.4. These rules mention flexible and rigid contexts, defined formally in section 2.4. The one-step instance relation $\dot{\sqsubseteq}$, which combines $\dot{\sqsubset}$ and $\dot{\sqsubset\!\!\!\!-}$ is defined in Figure 2.5. The symbol $\mathcal{R}$ stands either for $\dot{\sqsubseteq}$ or $\dot{\sqsubset\!\!\!\!-}^{\emptyset}$. As usual, we write $\sigma_1\ \mathcal{R}\ \sigma_2$ for $(Q)\ \sigma_1\ \mathcal{R}\ \sigma_2$, where Q is unconstrained and σ_1 and σ_2 are closed under Q.

Definition 2.6.1 We define the relations $(\equiv\dot{\sqsubset\!\!\!\!-}^{\emptyset})$ and $(\equiv\dot{\sqsubseteq})$: $(Q)\ \sigma_1\ (\equiv\mathcal{R})\ \sigma_2$ holds if and only if $(Q)\ \sigma_1 \equiv \sigma_2$ or there exists σ_1' and σ_2' such that $(Q)\ \sigma_1 \equiv \sigma_1'$, $(Q)\ \sigma_1'\ \mathcal{R}\ \sigma_2'$ and $(Q)\ \sigma_2' \equiv \sigma_2$. □

As already mentioned, $\mathcal{R}^*$ is the transitive closure of the relation $\mathcal{R}$. Note that StSh-Hyp is restricted to variables α not in $\bar{\alpha}$. The only rule introducing variables in $\bar{\alpha}$ is C-Abstract-F, which adds $\overline{C_f}$.

Figure 2.4: Strict Abstraction Relation

In all rules, C_r is a well-formed useful context.

StSh-Hyp

$$\frac{\begin{array}{c}\sigma' \notin \mathcal{T} \\ (\alpha = \sigma_1) \in Q \qquad \alpha \notin \mathsf{dom}(C_r) \qquad \alpha \notin \bar{\alpha} \qquad (Q)\ \sigma_1 \equiv \widehat{C_r}(\sigma')\end{array}}{(Q)\ C_r(\sigma')\ \mathrel{\dot{\sqsubset}\!\!\!\!-}^{\bar{\alpha}}\ C_r(\alpha)}$$

StSh-Up

$$\frac{\begin{array}{c}\beta \in \mathsf{ftv}(\sigma'') \qquad \alpha \in \mathsf{ftv}(\sigma') \qquad \sigma', \sigma'' \notin \mathcal{V} \qquad \sigma \notin \mathcal{T} \\ \alpha \notin \mathsf{dom}(Q\overline{C_R}Q') \qquad \mathsf{ftv}(\sigma)\ \#\ \mathsf{dom}(Q')\end{array}}{(Q)\ C_r(\forall\,(\beta = \forall\,(Q', \alpha = \sigma)\ \sigma')\ \sigma'')\ \mathrel{\dot{\sqsubset}\!\!\!\!-}^{\bar{\alpha}}\ C_r(\forall\,(\alpha = \sigma)\ \forall\,(\beta = \forall\,(Q')\ \sigma')\ \sigma'')}$$

StSh-Alias

$$\frac{\begin{array}{c}\alpha_1 \in \mathsf{ftv}(\sigma') \qquad \alpha_2 \in \mathsf{ftv}(\sigma') \qquad \mathsf{dom}(Q')\ \#\ \mathsf{ftv}(\sigma_1, \sigma_2) \qquad \sigma_1 \notin \mathcal{T} \\ \sigma = \forall\,(\alpha_1 = \sigma_1, Q', \alpha_2 = \sigma_2)\ \sigma' \qquad \sigma_1 \equiv \sigma_2\end{array}}{(Q)\ C_r(\sigma)\ \mathrel{\dot{\sqsubset}\!\!\!\!-}^{\bar{\alpha}}\ C_r(\forall\,(\alpha_1 = \sigma_1, Q', \alpha_2 = \alpha_1)\ \sigma')}$$

Figure 2.5: Combined instance

C-Strict

$$\frac{(Q)\ \sigma_1 \mathrel{\dot{\sqsubset}} \sigma_2}{(Q)\ \sigma_1 \mathrel{\dot{\sqsubseteq}} \sigma_2}$$

C-Abstract-F

$$\frac{(Q\overline{C_f})\ \sigma_1\ \mathrel{\dot{\sqsubset}\!\!\!\!-}^{\overline{C_f}}\ \sigma_2 \qquad \mathsf{level}(C_f) > 1}{(Q)\ C_f(\sigma_1) \mathrel{\dot{\sqsubseteq}} C_f(\sigma_2)}$$

C-Abstract-R

$$\frac{(Q)\ \sigma_1 \mathrel{\sqsubset\!\!\!\!-} \sigma_2}{(Q)\ \sigma_1 \mathrel{\dot{\sqsubseteq}} \sigma_2}$$

2.6.2 Equivalence between relations

These new relations have the same expressiveness as the original relations, as stated by the following lemma.

Properties 2.6.2

i) *The relations* $(\sqsubset\!\!\!\!-)$ *and* $(\equiv\mathrel{\dot{\sqsubset}\!\!\!\!-}^{\emptyset})^*$ *are equal.*

ii) *The relations* $(\sqsubseteq)$ *and* $(\equiv\dot{\sqsubseteq})^*$ *are equal.*

See proof in the Appendix (page 247).

The relations $\dot{\sqsubset}$ and $\mathrel{\dot{\sqsubset}\!\!\!\!-}^{\bar{\alpha}}$ can be seen as rewriting relations, under a given prefix. Then $(Q)\ \sigma_1 \mathrel{\dot{\sqsubset}} \sigma_2$ corresponds to reducing the argument σ_1 by $\dot{\sqsubset}$, and getting σ_2 as a result. Similarly, the relation $\dot{\sqsubseteq}^{\bar{\alpha}}$ is viewed as a rewriting relation. These (rewriting) relations have a remarkable property: if they can reduce an argument σ, then they can also reduce its normal form $\mathsf{nf}(\sigma)$. Besides, the results are equivalent. This is stated below:

Properties 2.6.3 *The following properties hold for $\mathcal{R}$ being $\dot{\sqsubseteq}^{\bar{\alpha}}$ and $\dot{\sqsubset}$.*

i) *If $(Q)\ \sigma_1\ \mathcal{R}\ \sigma_2$ holds, then there exists σ_2' such that $(Q)\ \mathsf{nf}(\sigma_1)\ \mathcal{R}\ \sigma_2'$ and $(Q)\ \sigma_2 \equiv \sigma_2'$ hold.*

ii) *If $(Q)\ \sigma_1\ \mathcal{R}\ \sigma_2$ holds, σ_1 is in normal form, and $\sigma_1' \approx \sigma_1$, then there exists σ_2' such that $(Q)\ \sigma_1'\ \mathcal{R}\ \sigma_2'$ and $(Q)\ \sigma_2 \equiv \sigma_2'$ hold.*

This is shown by considering each rule individually. In all cases, the projection of the redex to its normal form leads to a similar redex, thanks to Lemma 2.4.5 (page 75). See proof in the Appendix (page 250).

This result will be used in the proof of the main confluence result, namely the Diamond Lemma (Lemma 2.8.4).

2.7 Equivalence *vs* instantiation

The key result of this section is Property 2.7.7.i, which shows that the equivalence relation is the kernel of the instance relation. The second main result of this section is Corollary 2.7.10, which shows that both the abstraction relation and its inverse the revelation relation are stationary.

While occurrences describe only the symbolic structure of types, *i.e.* their skeleton, we here use polynomials to describe their binding structure. The polynomial associated to the binding structure of a type is called its weight. As expected, a type with no binding structure, *i.e.* a monotype, weighs zero, as stated by Lemma 2.7.4. The converse is also true.

We start by stating a few properties about polynomials, then we define weights, and show some useful properties. Then we prove the main results, which are Property 2.7.7.i and Corollary 2.7.10.

2.7.1 Polynomials

The set of three-variable polynomials in X, Y, Z, which we simply call "polynomials", is defined as $\mathbb{Z}[X][Y][Z]$. The addition $+$ and multiplication $\times$ are, as expected, commutative, thus, for instance $X \times Y$ equals $Y \times X$. It is well known [Bou59] that $\mathbb{Z}[X][Y][Z]$ is isomorphic to $(\mathbb{Z}[Y][Z])[X]$. In other words, any polynomial can be seen as a polynomial in variable X, with coefficients in the ring $\mathbb{Z}[Y][Z]$. Similarly, $\mathbb{Z}[Y][Z]$ can be seen as polynomials in variable Y, with coefficients in the ring $\mathbb{Z}[Z]$.

Given a total ordering $\leqslant$ on a ring $\mathbb{A}$, we define a total ordering on $\mathbb{A}\,[X]$ as the lexicographic order on the sequence of coefficients, higher degrees coming first. Formally, if P and Q are in $\mathbb{A}\,[X]$, we can always write them in the form $p_nX^n + \ldots + p_1X + p_0$ and $q_nX^n + \ldots + q_1X + q_0$. (we can have $p_n = 0$ if, for instance, the degree of P is smaller than the degree of Q). Then $P \leqslant Q$ if and only if $(p_n, \ldots, p_1, p_0) \leqslant (q_n, \ldots, q_1, q_0)$, taking the lexicographic order over $\mathbb{A}^{\,n+1}$.

Using the usual total ordering on $\mathbb{Z}$, we get a total ordering on $\mathbb{Z}[Z]$. We use again this ordering to build a total ordering on $(\mathbb{Z}[Z])[Y]$. This new ordering is used too to build a total ordering on $(\mathbb{Z}[Z][Y])[X]$.

Because of the above hierarchy, polynomial variables do not play equivalent roles, with respect to the orderings. More precisely, X is prevailing over Y which is in turn prevailing over Z.

Example The following sequence is an increasing sequence: Z, Z^2, Y, X, $X + Y^3$, X^2, Y^2X^2, $X^3 + Z$, X^4.

Intuitively, each binder of a type will be associated a polynomial variable (X, Y, or Z). Whereas the abstraction relation only modifies Y and Z binders, irreversible instantiations modify X binders. The following definition make it easier to talk about the X variables of a given polynomial.

Definition 2.7.1 The X-degree of a polynomial P is its degree in the variable X, where the coefficients are considered in the ring $(\mathbb{Z}[Z][Y])$. We write $X \in P$ if the X-degree of P is not 0. As a matter of fact, we have $X \in P$ if and only if X appears in the reduced form of P. □

This definition is used in the first and fourth properties of the following. The other results are straightfoward. We omit the proof.

Properties 2.7.2 *We have the following properties for any P, Q and R in $\mathbb{Z}[X][Y][Z]$:*

- ***i)*** *If $X \notin P$ and $X \notin P'$, then $X \notin P + P'$.*
- ***ii)*** *If $P \geqslant Q$, then $P + R \geqslant Q + R$.*
- ***iii)*** *If $P > 0$ and $R > 0$, then $P \times R > 0$.*
- ***iv)*** *If we have $P \leqslant Q \leqslant R$ and $X \notin R - P$, then $X \notin R - Q$ and $X \notin Q - P$.*

The latter property can be intuitively interpreted as follows: consider three types σ_p, σ_q and σ_r such that $\sigma_p \sqsubseteq \sigma_q$ $(\underline{\mathbf{1}})$ and $\sigma_q \sqsubseteq \sigma_r$ $(\underline{\mathbf{2}})$ hold (under some unspecified prefix). We write P, Q and R for the polynomials associated to σ_p, σ_q and σ_r, respectively. If some X binder is modified during the first instantiation (1) or during the second one (2), which implies $X \in Q - P$ or $X \in R - Q$, then X is to appear in $R - P$. In other words, each transformation on some X binder during instantiation will remain visible forever.

2.7.2 Weight

If $(Q)\ \sigma_1 \equiv \sigma_2$ holds, then we have $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ and $(Q)\ \sigma_2 \sqsubseteq \sigma_1$ by I-EQUIV* and symmetry of $\equiv$. Conversely, we want to show that if both $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ and $(Q)\ \sigma_2 \sqsubseteq \sigma_1$ hold, then $(Q)\ \sigma_1 \equiv \sigma_2$ is derivable. As mentioned above, each type is given a weight which reflects its binding structure. While weights are stable under equivalence, they strictly decrease by irreversible instantiation and irreversible abstraction. In the following paragraph, we give an illustrating example that should give the intuition behind weights. The definitions or terms that we introduce there are not meant to be formal and are not used further in the document.

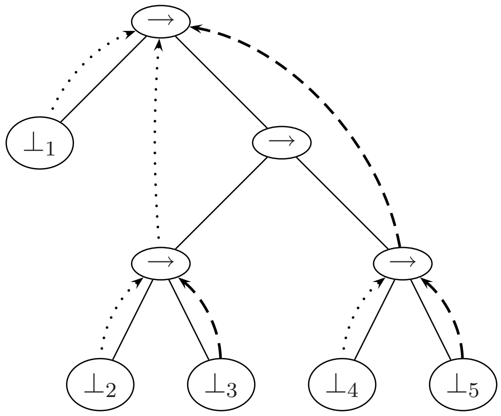
Introduction to weights Weights are meant to give some information about the binding structure of types. For example, take the following type, written σ (the type being too large, we display it on several lines)

$$\begin{array}{l}
\forall\,(\alpha \geq \bot) \\
\forall\,(\beta \geq \forall\,(\beta_1 \geq \bot)\ \forall\,(\beta_2 = \bot)\ \beta_1 \to \beta_2) \\
\forall\,(\gamma = \forall\,(\gamma_1 \geq \bot)\ \forall\,(\gamma_2 = \bot)\ \gamma_1 \to \gamma_2) \\
\alpha \to \beta \to \gamma
\end{array}$$

Its binding structure depends only on the locations of $\bot$. In the above example, $\bot$ occurs five times. For illustration purposes, we give a number to each occurrence of $\bot$:

$$\begin{array}{l}
\forall\,(\alpha \geq \bot_1) \\
\forall\,(\beta \geq \forall\,(\beta_1 \geq \bot_2)\ \forall\,(\beta_2 = \bot_3)\ \beta_1 \to \beta_2) \\
\forall\,(\gamma = \forall\,(\gamma_1 \geq \bot_4)\ \forall\,(\gamma_2 = \bot_5)\ \gamma_1 \to \gamma_2) \\
\alpha \to \beta \to \gamma
\end{array}$$

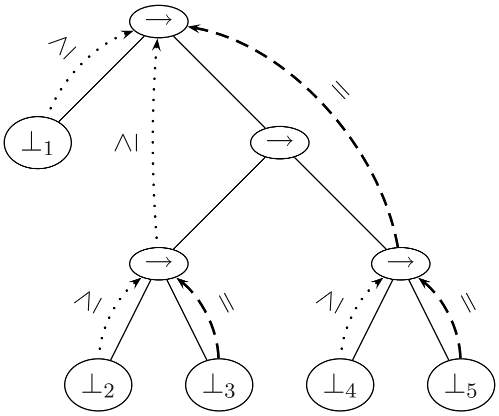
We represent this type with the following graph:



We consider all occurrences of $\bot$ and (informally) associate a *path* to each occurrence. In this context, a path is a word in the set $\{=, \geq\}$, such as $\geq \geq = \geq \geq$.

- $\bot_1$ appears inside a flexible context (see page 74) that is directly in the bound of α, which is flexible. Hence, the path associated to $\bot_1$ is $\geq$.
- $\bot_2$ appears directly in the bound of β_1, and β_1 is in the bound of β. Both β and β_1 are flexible, thus the path associated to $\bot_2$ is written $\geq\geq$.
- $\bot_3$ appears in the bound of β_2, which is rigid, and β_2 is in the bound of β, which is flexible. Hence, the path associated to $\bot_3$ is $\geq=$.
- Similarly, the path associated to $\bot_4$ is $=\geq$, and the path associated to $\bot_5$ is $==$.

On graphs, we simply label flexible bindings with $\geq$ and rigid bindings with $=$.



We now show in detail how we compute the polynomial associated to this type. The first step consists in associating a polynomial variable to each binding arrow. Then, each path can be easily mapped to a monomial. The second step simply adds all the monomials associated to the paths of occurrences of $\bot$.

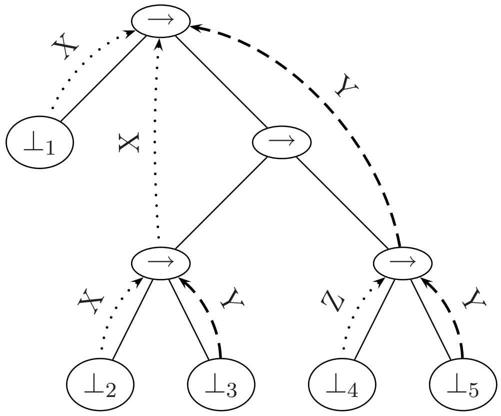
- Both $\bot_1$ and $\bot_2$ appear in flexible contexts (see page 74), that is, their path is of the form $\geq^*$ (a sequence of $\geq$). Instantiation ($\sqsubseteq$) is allowed in such contexts. We will weight such contexts using the polynomial variable X. More precisely, the weight associated to $\geq$ (that is, to the path of $\bot_1$) is X, and the weight associated to $\geq\geq$ (that is, to the path of $\bot_2$) is XX, that is X^2.
- Both $\bot_3$ and $\bot_5$ appear in rigid contexts (see page 74), that is, their path is of the form $\geq^*=^+$ (a sequence of $\geq$ followed by a non-empty sequence of $=$). Abstraction

($\sqsubseteq$) is allowed in such contexts, but not instantiation. We will weight the rigid part of such contexts using the polynomial variable Y. More precisely, the weight associated to $\geq=$ (that is, the path of $\perp_3$) is XY, and the weight associated to $==$ (that is, the weight of $\perp_5$) is YY, that is Y^2.

- Finally, the path associated to $\perp_4$ is $=\geq$. The context of $\perp_4$ is of the form $C_r(C_f)$, and neither abstraction nor instantiation is possible in such a context. We will weight the non-flexible and non-rigid part of such contexts using the polynomial variable Z. More precisely, the weight associated to $=\geq$ (that is, the path of $\perp_4$) is YZ.

The total weight of σ is obtained by adding all the monomials computed for each occurrence of $\perp$. Thus, we get the polynomial $X + X^2 + XY + Y^2 + YZ$.

We can also label each binder of the graph above with its corresponding weight.



In summary, we use polynomial variables to weigh the contexts in which $\perp$ occurs. The polynomial variable X is used for flexible contexts whose path is of the form $\geq^i$, the polynomial variable Y is used for rigid contexts whose path is of the form $\geq^i=^{j+1}$ and the polynomial variable Z is used for contexts whose path if of the form $\geq^i =^{j+1} \geq\diamond^k$. This is summarized in Figure 2.6. We wish to define weight incrementally. Hence, we assume given a “current” path that we incrementally extend. Assume the current path is $\geq^i$. It weighs X^i, and its associated variable is X. If we add $\geq$ to this path, we see that the weight is multiplied by X, which gives X^{i+1}. If we add $=$, the weight is multiplied by Y, which gives X^iY. This is why we define the auxiliary operator $\star$ below, such that $X \star \geq$ is X and $X \star =$ is Y. To continue, assume the current path is $\geq^i=^{j+1}$. It weighs X^iY^{j+1}, and its associated variable is Y. If we add $\geq$ to this path, the weight is multiplied by Z, which gives $X^iY^{j+1}Z$. If we add $=$, the weight is multiplied by Y. Hence, we define $Y \star \geq$ as Z and $Y \star =$ as Y. Similarly, if the

Figure 2.6: Paths and weights

Path	Associated polynomial variable
$\underbrace{\overbrace{\geq \ldots \geq}^{i \text{ times}}}_{X^i}$	X
$\underbrace{\overbrace{\geq \ldots \geq}^{i \text{ times}}}_{X^i} \quad \underbrace{\overbrace{= \ldots =}^{j+1 \text{ times}}}_{Y^{j+1}}$	Y
$\underbrace{\overbrace{\geq \ldots \geq}^{i \text{ times}}}_{X^i} \quad \underbrace{\overbrace{= \ldots =}^{j+1 \text{ times}}}_{Y^{j+1}} \quad \underbrace{\overbrace{\geq \diamond_1 \ldots \diamond_k}^{k \text{ times}}}_{Z^{k+1}}$	Z

current path is $\geq^i =^{j+1} \geq \diamond^k$, the associated variable is Z. Adding X or Y to such a path multiplies the weight by Z. Hence, $Z \star \geq$ and $Z \star =$ are both equal to Z. This should explain the following definition:

Definition 2.7.3 We define the auxiliary binary operator $\star$ on the set $\{X, Y, Z\} \times \{\geq, =\}$ as follows:

$$X \star \geq \text{ is } X \qquad X \star = \text{ is } Y \qquad Y \star \geq \text{ is } Z \qquad Y \star = \text{ is } Y \qquad Z \star \diamond \text{ is } Z \qquad \square$$

As seen above, the weight associated to each occurrence of $\perp$ depends on the its path, that is, on its context. The polynomial variables X, Y, and Z represent three kinds of paths. This is why we define three functions w_A for A in $\{X, Y, Z\}$, where A represents intuitively the current path. Each w_A maps types to polynomials in $\mathbb{Z}[X][Y][Z]$. Here is the structural definition of w_A:

$$w_A(\perp) \triangleq 1 \qquad\qquad w_A(\tau) \triangleq 0$$

$$w_A(\forall\,(\alpha \diamond \sigma)\ \sigma') \triangleq \begin{cases} w_A(\sigma') & \text{if } \alpha \notin \mathsf{ftv}(\sigma') \\ w_A(\sigma) & \text{if } \mathsf{nf}(\sigma') = \alpha \\ w_{A \star \diamond}(\sigma) \times (A \star \diamond) + w_A(\sigma') & \text{otherwise} \end{cases}$$

By default, the notation $w(\sigma)$ means $w_X(\sigma)$. Note that $w_A(\sigma) \geqslant 0$ for any A and σ, that is, $w_A(\sigma)$ is in $I\!N[X][Y][Z]$.

Examples One can check that σ, given above, does weigh $X+X^2+XY+Y^2+YZ$. As another example, the weight of $\forall\,(\beta=\forall\,(\alpha\geq\bot)\ \alpha\rightarrow\alpha)\ \beta\rightarrow\beta$ is Z^2 and the weight of the type of the identity $\forall\,(\alpha\geq\bot)\ \alpha\rightarrow\alpha$ is X. Note that $\forall\,(\alpha\geq\bot)\ \alpha\rightarrow\alpha$ and $\forall\,(\alpha\geq\bot)\ (\alpha\rightarrow\alpha)\rightarrow(\alpha\rightarrow\alpha)$ have the same weight, whereas their structure (skeleton) are different.

We have characterized monotypes by their normal form, or by their projection (as in Property 2.1.5.i (page 67)). Another way is to use weights:

Lemma 2.7.4 *We have $w_A(\sigma)=0$ iff $\sigma\in\mathcal{T}$.*

Proof: By Property 2.1.5.i (page 67), we only have to show that $w_A(\sigma)=0$ if and only if there is no u such that $\sigma/u=\bot$. The proof by structural induction on σ is straightforward. ■

Understanding weight

Let's have a look at the rules defining abstraction $\sqsubseteq$ and instantiation $\sqsubseteq$. We can consider these rules as rewriting rules taking a type and returning an instantiated type. Rules such as R-Context-R, R-Context-Flexible, … define the contexts in which the rewriting can be applied. Some rules are not symmetric, and possibly not reversible, namely the rules A-Hyp, I-Hyp, I-Rigid and I-Bot. Each time I-Bot is used (with restrictions of Lemma 2.3.1), the domain of the projection of its argument grows: if $(Q)\ \sigma\sqsubseteq\sigma'$ holds by I-Bot and context rules, then the domain of σ is included in the domain σ'. For example, consider $\bot\sqsubseteq\alpha\rightarrow\alpha$: the domain grows from $\{\epsilon\}$ to $\{\epsilon,1,2\}$. However, rules A-Hyp, I-Hyp and I-Rigid keep the same projection (relative to a given prefix). This is immediate for I-Rigid since projections ignore the kind of bounds. As for A-Hyp (or, similarly, I-Hyp), consider this typical example:

$$(\alpha=\sigma)\ \forall\,(\beta=\sigma)\ \beta\rightarrow\alpha\sqsubseteq\forall\,(\beta=\alpha)\ \beta\rightarrow\alpha$$

It can be derived by R-Context-Rigid and A-Hyp. The skeleton associated to both sides (and under the prefix $(\alpha=\sigma)$) is $\mathsf{proj}(\sigma)\rightarrow\mathsf{proj}(\sigma)$. Hence, in such an example, the projection is kept unchanged. Binders, however, have been modified, and weights are modified too. Indeed, the weight of $\forall\,(\beta=\sigma)\ \beta\rightarrow\alpha$ is $Y\times w_Y(\sigma)$, while the weight of $\forall\,(\beta=\alpha)\ \beta\rightarrow\alpha$ is 0. We see that weights and projections are complementary: although projections are stable under A-Hyp, I-Hyp and I-Rigid, these rules decrease weights. However, we do not have $(Q)\ \sigma_1\equiv\sigma_2$ if and only if $w(\sigma_1)=w(\sigma_2)$ and $\forall\,(Q)\ \sigma_1/=\forall\,(Q)\ \sigma_2/$. This result is true only if σ_2 is known to be an instance of σ_1 under Q, as stated in Property 2.7.6.ii. As a counterexample, take $\sigma_1\triangleq\forall\,(\alpha\geq\bot)\ \forall\,(\beta\geq\sigma_{\mathtt{id}})\ (\alpha\rightarrow\alpha)\rightarrow\beta$ and $\sigma_2\triangleq\forall\,(\alpha\geq\bot)\ \forall\,(\beta\geq\sigma_{\mathtt{id}})\ \beta\rightarrow(\alpha\rightarrow\alpha)$, to be taken under an empty prefix. We do have $\sigma_1 / = \sigma_2 /$, more precisely both σ_1 and σ_2 have the skeleton $(\bot \rightarrow \bot) \rightarrow (\bot \rightarrow \bot)$. Besides, $w(\sigma_1)$ and $w(\sigma_2)$ are both equal to $X + X^2$. However, σ_1 and σ_2 are not equivalent since they do not have the same normal form up to rearrangement.

Like projections, weights are stable under equivalence:

Lemma 2.7.5 *If* $(Q)\ \sigma_1 \equiv \sigma_2$ *holds, then* $w_A(\sigma_1) = w_A(\sigma_2)$.

Proof: By induction on the derivation of $(Q)\ \sigma_1 \equiv \sigma_2$. Case Eq-Refl is obvious. Cases Eq-Comm, Eq-Var, and Eq-Free are immediate by definition of w_A. In the case Eq-Mono, both sides weigh 0. Cases R-Trans and R-Context-L are by induction hypothesis. As for R-Context-R, we have $\sigma_1 = \forall\,(\alpha \diamond \sigma)\ \sigma_1'$, $\sigma_2 = \forall\,(\alpha \diamond \sigma)\ \sigma_2'$, and the premise is $(Q, \forall\,(\alpha \diamond \sigma))\ \sigma_1' \equiv \sigma_2'$ **(<u>1</u>)** We have to show that $w_A(\forall\,(\alpha \diamond \sigma)\ \sigma_1') = w_A(\forall\,(\alpha \diamond \sigma)\ \sigma_2')$. By induction hypothesis and (1), we have $w_{A'}(\sigma_1') = w_{A'}(\sigma_2')$ **(<u>2</u>)**, for any A' in $\{X, Y, Z\}$. We proceed by case analysis:

Subcase $\sigma \in \mathcal{T}$: Then by Lemma 2.7.4 we have $w_A(\sigma) = 0$, thus we have $w_A(\forall\,(\alpha \diamond \sigma)\ \sigma_1') = w_A(\sigma_1')$ and $w_A(\forall\,(\alpha \diamond \sigma)\ \sigma_2') = w_A(\sigma_2')$, and the result follows from (2). We can now consider σ not in $\mathcal{T}$, that is $\alpha \notin \mathsf{dom}(\widehat{Q, \alpha \diamond \sigma})$. Writing θ for $\widehat{Q, \alpha \diamond \sigma}$, we have $\alpha \notin \mathsf{dom}(\theta)$ **(<u>3</u>)**.

Subcase $\alpha \notin \mathsf{ftv}(\sigma_1')$: Then by Property 1.5.11.ix (page 54), (1), and (3), we have $\alpha \notin \mathsf{ftv}(\sigma_2')$. Hence, we have $w_A(\forall\,(\alpha \diamond \sigma)\ \sigma_1') = w_A(\sigma_1')$, $w_A(\forall\,(\alpha \diamond \sigma)\ \sigma_2') = w_A(\sigma_2')$, and we conclude by (2).

Subcase $\alpha \notin \mathsf{ftv}(\sigma_2')$: similar. We consider now that $\alpha \in \mathsf{ftv}(\sigma_1')$ and $\alpha \in \mathsf{ftv}(\sigma_2')$.

Subcase $\mathsf{nf}(\sigma_1') = \alpha$: Then $\theta(\alpha) = \theta(\mathsf{nf}(\sigma_2'))$ holds from Lemma 1.5.9 and (1). Hence, we get $\alpha = \theta(\mathsf{nf}(\sigma_2'))$ **(<u>4</u>)** from (3). By well-formedness of $(Q, \alpha \diamond \sigma)$, we have $\alpha \notin \mathsf{ftv}(\widehat{Q, \alpha \diamond \sigma})$, thus we have $\alpha \notin \mathsf{codom}(\theta)$. Hence (4) gives $\mathsf{nf}(\sigma_2') = \alpha$. Thus $w_A(\forall\,(\alpha \diamond \sigma)\ \sigma_1') = w_A(\sigma) = w_A(\forall\,(\alpha \diamond \sigma)\ \sigma_2')$, which is the expected result.

Subcase $\mathsf{nf}(\sigma_2') = \alpha$: similar.

Otherwise we have $\mathsf{nf}(\sigma_1') \neq \alpha$, $\mathsf{nf}(\sigma_2') \neq \alpha$, $\alpha \in \mathsf{ftv}(\sigma_1')$, and $\alpha \in \mathsf{ftv}(\sigma_2')$, hence by definition we get $w_A(\forall\,(\alpha \diamond \sigma)\ \sigma_1') = w_{A \star \diamond}(\sigma) \times (A \star \diamond) + w_A(\sigma_1')$ and $w_A(\forall\,(\alpha \diamond \sigma)\ \sigma_2') = w_{A \star \diamond}(\sigma) \times (A \star \diamond) + w_A(\sigma_2')$. They are equal by (2). ■

How to lose weight

Although weights are stable under equivalence, instantiation decreases weight and increases the domain of projections.

Properties 2.7.6 *If $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!\!-} \sigma_2$ holds, then the two following properties are true for $A = Y$ and $A = X$. If $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ holds, then the two following properties are true for $A = X$.*

- ***i)*** *If $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$, then $w_A(\sigma_1) \geqslant w_A(\sigma_2)$.*
- ***ii)*** *If $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$ and $w_A(\sigma_1) = w_A(\sigma_2)$, then $(Q)\ \sigma_1 \equiv \sigma_2$ holds.*

Proof: We prove both properties simultaneously. By Lemma 2.5.5, we have a derivation of $(Q)\ \sigma_1 \Diamond^{\bar\alpha} \sigma_2$ **(1)**. By Lemma 2.5.6, the derivation is assumed restricted. We prove both properties by induction on the derivation of (1).

∘ Case A-Equiv': We have $(Q)\ \sigma_1 \equiv \sigma_2$. By Lemma 2.7.5, we have $w_A(\sigma_1) = w_A(\sigma_2)$. Hence, Properties i and ii hold.

∘ Case R-Trans The premises are $(Q)\ \sigma_1 \Diamond^{\bar\alpha} \sigma_1'$ **(2)** and $(Q)\ \sigma_1' \Diamond^{\bar\alpha} \sigma_2$ **(3)**. By induction hypothesis on (2), if $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_1'/$, then $w_A(\sigma_1) \geqslant w_A(\sigma_1')$ **(4)**. By induction hypothesis on (3), if $\forall\,(Q)\ \sigma_1'/ = \forall\,(Q)\ \sigma_2/$, then $w_A(\sigma_1') \geqslant w_A(\sigma_2)$ **(5)**. By Property 2.1.3.ii (page 65), (2), and (3), we have $\forall\,(Q)\ \sigma_1 \leqslant_{/} \forall\,(Q)\ \sigma_1'$ **(6)**, and $\forall\,(Q)\ \sigma_1' \leqslant_{/} \forall\,(Q)\ \sigma_2$ **(7)**. We prove Property i: Assume $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$ holds. Then we have $\forall\,(Q)\ \sigma_2 \leqslant_{/} \forall\,(Q)\ \sigma_1'$ from (6) thus $\forall\,(Q)\ \sigma_1'/ = \forall\,(Q)\ \sigma_2/ = \forall\,(Q)\ \sigma_1/$ holds by antisymmetry (Property 2.1.2.i (page 65)) and (7). As a consequence, we have $w_A(\sigma_1) \geqslant w_A(\sigma_1')$ **(8)** from (4) and $w_A(\sigma_1') \geqslant w_A(\sigma_2)$ **(9)** from (5). Thus, we have $w_A(\sigma_1) \geqslant w_A(\sigma_2)$ by transitivity of $\geqslant$ on polynomials. This proves Property i.

By induction hypothesis on (2), if $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_1'/$, and $w_A(\sigma_1) = w_A(\sigma_1')$, then there exists a derivation of $(Q)\ \sigma_1 \equiv \sigma_1'$ **(10)**. By induction hypothesis on (3), if $\forall\,(Q)\ \sigma_1'/ = \forall\,(Q)\ \sigma_2/$, and $w_A(\sigma_1') = w_A(\sigma_2)$, then there exists a derivation of $(Q)\ \sigma_1' \equiv \sigma_2$ **(11)**. We prove Property ii: As seen above, if $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$, then $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/ = \forall\,(Q)\ \sigma_1'/$. Additionally, if $w_A(\sigma_1) = w_A(\sigma_2)$, then (8) and (9) imply $w_A(\sigma_1) = w_A(\sigma_1') = w_A(\sigma_2)$, by antisymmetry of $\geqslant$ on polynomials. Consequently, we have a derivation of $(Q)\ \sigma_1 \equiv \sigma_1'$ by (10) and a derivation of $(Q)\ \sigma_1' \equiv \sigma_2$ by (11). Hence, we get a derivation of $(Q)\ \sigma_1 \equiv \sigma_2$ by R-Trans. This proves Property ii.

∘ Case A-Hyp' and I-Hyp': We have either $(Q)\ \sigma_1 \sqsubseteq \alpha_1$ (that is, σ_2 is α_1), with $(\alpha_1 \geq \sigma_1) \in Q$, or $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!\!-} \alpha_1$ with $(\alpha_1 = \sigma_1) \in Q$. By Lemma 2.3.1, σ_1 is not in $\mathcal{T}$, thus $w_A(\sigma_1) \neq 0$ by Lemma 2.7.4. Additionally, $w_A(\sigma_2) = w_A(\alpha_1) = 0$ by definition. Hence, $w_A(\sigma_1) > w_A(\sigma_2)$, which proves Properties i and ii.

∘ Case A-Alias' and I-Alias': We have $\sigma_1 = \forall\,(\alpha_1 \diamond \sigma)\ \forall\,(\alpha_2 \diamond \sigma)\ \sigma'$ and $\sigma_2 = \forall\,(\alpha_1 \diamond \sigma)\ \forall\,(\alpha_2 = \alpha_1)\ \sigma'$. By restrictions of Lemma 2.5.6, we have $\alpha_1 \in \mathsf{ftv}(\sigma')$ and $\alpha_2 \in \mathsf{ftv}(\sigma')$. Besides, $\sigma \notin \mathcal{T}$ **(1)**. By definition we have $w_A(\sigma_1) = w_A(\sigma') + 2 \times B \times w_B(\sigma)$, where B is $A{\star}\diamond$, and we have $w_A(\sigma_2) = w_A(\sigma') + B \times w_B(\sigma)$. Hence, $w_A(\sigma_1) - w_A(\sigma_2) = B \times w_B(\sigma)$, which is strictly greater than 0 by Lemma 2.7.4 and (1). This implies $w_A(\sigma_1) > w_A(\sigma_2)$ by Property 2.7.2.ii (page 87), thus Properties i and ii hold.

∘ Case A-Up' and I-Up': We have $\sigma_1 = \forall\,(\alpha \diamond \forall\,(\alpha' \diamond' \sigma')\ \sigma)\ \sigma''$ and $\sigma_2 = \forall\,(\alpha' \diamond' \sigma')\ \forall\,(\alpha \diamond \sigma)\ \sigma''$. We have $w_A(\sigma_1) = w_A(\sigma'') + B \times (w_B(\sigma) + C \times w_C(\sigma'))$ **(1)**, where B is $A{\star}\diamond$ and C is $B{\star}\diamond'$. Similarly, we have $w_A(\sigma_2) = w_A(\sigma'') + D \times w_D(\sigma') + B \times w_B(\sigma)$ **(2)**

where D is $A \star \diamond'$. If we are in the case A-Up', then $\diamond$ and $\diamond'$ are rigid, thus B is $A\star =$, that is Y (A is X or Y), C is Y, and D is Y. If we are in the case I-Up', then $\diamond$ is $>$, A is X, thus B is X, C is $X \star \diamond'$, and D is $X \star \diamond'$. In both cases, $C = D$. Hence, we have $w_A(\sigma_2) = w_A(\sigma'') + C \times w_C(\sigma') + B \times w_B(\sigma)$ **($\underline{3}$)** from (2). From (1) and (3), we get $w_A(\sigma_1) - w_A(\sigma_2) = C \times w_C(\sigma') \times (B - 1)$ **($\underline{4}$)**. Since B is X or Y, we have $B > 1$, that is, $B - 1 > 0$ **($\underline{5}$)**. Since σ' is not in $\mathcal{T}$, we have $w_C(\sigma') > 0$ **($\underline{6}$)** by Lemma 2.7.4. Hence $w_A(\sigma_1) - w_A(\sigma_2) > 0$ by Property 2.7.2.iii (page 87), (5), (6), and (4), that is, $w_A(\sigma_1) > w_A(\sigma_2)$. This proves Properties i and ii.

◦ Case R-Context-R We have $(Q)\ \sigma_1 \Diamond^{\bar\alpha} \sigma_2$, with $\sigma_1 = \forall\,(\alpha \diamond \sigma_0)\ \sigma'_1$, $\sigma_2 = \forall\,(\alpha \diamond \sigma_0)\ \sigma'_2$, and the premise is $(Q, \alpha \diamond \sigma_0)\ \sigma'_1 \Diamond^{\bar\alpha \cup \{\alpha\}} \sigma'_2$ **($\underline{1}$)**. If $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$, then by notation $\forall\,(Q, \alpha \diamond \sigma_0)\ \sigma'_1/ = \forall\,(Q, \alpha \diamond \sigma_0)\ \sigma'_2/$, thus, by induction hypothesis, Property i, and (1), we have $w_A(\sigma'_1) \geqslant w_A(\sigma'_2)$ **($\underline{2}$)**. We proceed by case analysis.

Subcase $\sigma_0 \in \mathcal{T}$: Then by definition we have $w_A(\sigma_1) = w_A(\sigma'_1)$ and $w_A(\sigma_2) = w_A(\sigma'_2)$, thus (2) proves Property i. Additionally, if we have $w_A(\sigma_1) = w_A(\sigma_2)$, then we have $w_A(\sigma'_1) = w_A(\sigma'_2)$, thus $(Q, \alpha \diamond \sigma_0)\ \sigma'_1 \equiv \sigma'_2$ holds by induction hypothesis and (1). We get $(Q)\ \sigma_1 \equiv \sigma_2$ by R-Context-R, which proves Property ii. In the following, we assume $\sigma_0 \notin \mathcal{T}$ **($\underline{3}$)**.

Subcase $\mathsf{nf}(\sigma'_1) = \alpha$: Then we have $(Q, \alpha \diamond \sigma_0)\ \sigma'_1 \equiv \sigma'_2$ by Lemma 2.1.6 and (1). Thus $(Q)\ \sigma_1 \equiv \sigma_2$ holds by R-Context-R, and we get the expected result by Lemma 2.7.5. In the following, we assume that $\mathsf{nf}(\sigma'_1)$ is not α.

Subcase $\mathsf{nf}(\sigma'_2) = \alpha$ and $\alpha \in \mathsf{ftv}(\sigma'_1)$ **($\underline{4}$)**: Then $\sigma'_1 \equiv \alpha$ by Property 2.1.7.ii (page 68) and (1), and this is the previous subcase.

Subcase $\alpha \in \mathsf{ftv}(\sigma'_1)$ and $\alpha \notin \mathsf{ftv}(\sigma'_2)$: This is not possible by Lemma 2.1.4 (page 67) and (3).

Subcase $\alpha \in \mathsf{ftv}(\sigma'_2)$ and $\alpha \notin \mathsf{ftv}(\sigma'_1)$ **($\underline{5}$)**: This is not possible by Lemma 2.5.7 (page 83) and (3) (α is in $\bar\alpha \cup \{\alpha\}$). In the following, we assume that $\mathsf{nf}(\sigma'_2)$ is not α (because such a case corresponds necessarily to subcases (4) or (5)).

Subcase $\alpha \in \mathsf{ftv}(\sigma'_2)$ and $\alpha \in \mathsf{ftv}(\sigma'_1)$: We assumed that $\mathsf{nf}(\sigma'_1) \neq \alpha$ and $\mathsf{nf}(\sigma'_2) \neq \alpha$, thus by definition, we have $w_A(\sigma_1) = w_B(\sigma_0) \times B + w_A(\sigma'_1)$, where B is $A \star \diamond$. Similarly, we have $w_A(\sigma_2) = w_B(\sigma_0) \times B + w_A(\sigma'_2)$. Hence $w_A(\sigma_1) - w_A(\sigma_2) = w_A(\sigma'_1) - w_A(\sigma'_2)$ **($\underline{6}$)** which is greater or equal to 0 by (2). This proves Property i. If $w_A(\sigma_1) = w_A(\sigma_2)$, then (6) implies $w_A(\sigma'_1) = w_A(\sigma'_2)$. Hence, by induction hypothesis and (1), there exists a derivation of $(Q, \alpha \diamond \sigma_0)\ \sigma'_1 \equiv \sigma'_2$. We get $(Q)\ \sigma_1 \equiv \sigma_2$ by R-Context-R. This proves Property ii.

◦ Case A-Context-L' and I-Context-L': We have $(Q)\ \sigma_1 \Diamond \sigma_2$, with $\sigma_1 = \forall\,(\alpha \diamond \sigma'_1)\ \sigma_0$ and $\sigma_2 = \forall\,(\alpha \diamond \sigma'_2)\ \sigma_0$. The premise is $(Q)\ \sigma'_1 \Diamond^{\bar\alpha} \sigma'_2$ **($\underline{1}$)**. Moreover, if $\Diamond^{\bar\alpha}$ is $\sqsubseteq^{\bar\alpha}$, then $\diamond$ is flexible; if $\Diamond^{\bar\alpha}$ is $\mathrel{-\!\!\!\sqsubset}^{\bar\alpha}$, then $\diamond$ is rigid. By Lemma 2.5.6, we have $\mathsf{nf}(\sigma_0) \neq \alpha$ and $\alpha \in \mathsf{ftv}(\sigma_0)$, that is, there exists u such that $\sigma_0/u = \alpha$ **($\underline{2}$)**. We prove Property i. Assume $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$ holds. Then by Property 1.3.3.i (page 40), we get $\Theta_Q(\sigma_1) = \Theta_Q(\sigma_2)$. Hence, we have $\Theta_Q(\sigma_1) \cdot u/ = \Theta_Q(\sigma_2) \cdot u/$, that is, $\Theta_Q(\sigma'_1) = \Theta_Q(\sigma'_2)$ by (2). By Property 1.3.3.i (page 40), this implies $\forall\,(Q)\ \sigma'_1/ = \forall\,(Q)\ \sigma'_2/$. Let B be $A \star \diamond$. By induction hypothesis and (1), we get $w_B(\sigma'_1) \geqslant w_B(\sigma'_2)$ **($\underline{3}$)**. We have

$w_A(\sigma_1) = w_A(\sigma_0) + w_B(\sigma_1') \times B$. Similarly, $w_A(\sigma_2) = w_A(\sigma_0) + w_B(\sigma_2') \times B$. Hence we have $w_A(\sigma_1) - w_A(\sigma_2) = B \times (w_B(\sigma_1') - w_B(\sigma_2'))$ **(4)**, thus we get $w_A(\sigma_1) \geqslant w_A(\sigma_2)$ from (3). This proves Property i. If $w_A(\sigma_1) = w_A(\sigma_2)$, then (4) implies $w_B(\sigma_1') = w_B(\sigma_2')$, thus $(Q)\ \sigma_1' \equiv \sigma_2'$ is derivable by induction hypothesis and (1). Hence $(Q)\ \sigma_1 \equiv \sigma_2$ is derivable by R-CONTEXT-L. This proves Property ii.

- CASE I-ABSTRACT': By induction hypothesis.

- CASE I-BOT': We have $\sigma_1 = \bot$. Hence, $(\forall\,(Q)\ \sigma_1)/\epsilon = \bot$. By Lemma 2.5.6, we have $\sigma_2 \notin \mathcal{V}$, and σ_2 is not $\bot$. Hence, σ_2/ϵ is a type constructor g. By definition, we have $\forall\,(Q)\ \sigma_1 <_{/} \forall\,(Q)\ \sigma_2$, which proves Properties i and ii.

- CASE I-RIGID': In this case, $\diamondsuit^{\bar\alpha}$ is $\sqsubseteq^{\bar\alpha}$, and A is X. We have $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$ and σ_1 is of the form $\forall\,(\alpha \geq \sigma)\ \sigma'$, while σ_2 is of the form $\forall\,(\alpha = \sigma)\ \sigma'$. By Lemma 2.5.6, we have $\alpha \in \mathsf{ftv}(\sigma')$, $\mathsf{nf}(\sigma') \neq \alpha$, and $\sigma \notin \mathcal{T}$ **(1)**. By definition, $w_X(\sigma_1)$ is $w_X(\sigma') + X \times w_X(\sigma)$, while $w_X(\sigma_2)$ equals $w_X(\sigma') + Y \times w_Y(\sigma)$. We get $w(\sigma_1) - w(\sigma_2) = Xw_X(\sigma) - Yw_Y(\sigma)$. By (1) and Lemma 2.7.4, we get $w_X(\sigma) \neq 0$. The X-degree of $Xw_X(\sigma)$ is at least 1, but the X-degree of $Yw_Y(\sigma)$ is 0. Hence, $Xw_X(\sigma) - Yw_Y(\sigma) > 0$, that is, $w(\sigma_1) > w(\sigma_2)$. This proves Properties i and ii. ■

Then we can show that the equivalence relation is the symmetric kernel of the instance relation.

Properties 2.7.7
i) *If we have* $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ *and* $(Q)\ \sigma_2 \sqsubseteq \sigma_1$*, then* $(Q)\ \sigma_1 \equiv \sigma_2$ *holds.*
ii) *If* $(Q)\ \sigma \sqsubseteq \bot$ *holds, then* $\mathsf{nf}(\sigma)$ *is* $\bot$*.*

Proof: Property i: By Property 2.1.3.ii (page 65), we get $\forall\,(Q)\ \sigma_1/ \leqslant_{/} \forall\,(Q)\ \sigma_2/$ and $\forall\,(Q)\ \sigma_2/ \leqslant_{/} \forall\,(Q)\ \sigma_1/$. Hence, by antisymmetry (Property 2.1.2.i (page 65)), we get $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$. By Property 2.7.6.i, we get $w(\sigma_1) \geqslant w(\sigma_2)$ and $w(\sigma_2) \geqslant w(\sigma_1)$. Hence, $w(\sigma_1) = w(\sigma_2)$, thus, by Property 2.7.6.ii, there exists a derivation of $(Q)\ \sigma_1 \equiv \sigma_2$.

Property ii: By I-BOT, we have $(Q)\ \bot \sqsubseteq \sigma$. By Property i, we get $(Q)\ \sigma \equiv \bot$. By Lemma 1.5.9, we get $\mathsf{nf}(\widehat{Q}(\sigma)) = \bot$. By Property 1.5.6.iii (page 51), this gives $\widehat{Q}(\mathsf{nf}(\sigma)) = \bot$. Hence, we must have $\mathsf{nf}(\sigma) = \bot$. ■

Weight watchers

As mentioned above, X corresponds intuitively to flexible contexts. Besides, the abstraction relation only occurs in rigid contexts. Thus, we expect "X-binders" to be kept unchanged by abstraction. Conversely, if an instantiation does not modify "X-binders", we expect it to be a true abstraction. This last point is stated exactly in the following lemma:

Lemma 2.7.8 *Assume* $\widehat{Q}(\mathsf{nf}(\sigma_2)) \notin \vartheta$. *Then we have* $(Q)\ \sigma_1 \sqsubseteq \sigma_2$, $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$, *and* $X \notin w(\sigma_1) - w(\sigma_2)$ *if and only if* $(Q)\ \sigma_1 \sqsubseteq\!\!\!\!-\ \sigma_2$.

It is shown by induction on the derivation of $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ for the first implication and by induction on the derivation of $(Q)\ \sigma_1 \sqsubseteq\!\!\!\!-\ \sigma_2$ for the second.
See details in the Appendix (page 253).

Note that "X-binders are kept unchanged" is expressed formally by $X \notin w(\sigma_1) - w(\sigma_2)$.

2.7.3 Abstraction is well-founded

In this section we show that abstraction is well-founded, as stated formally by Corollary 2.7.10. First, we show that the weight associated to a type σ is bounded by a polynomial that depends only on the size of σ.

Preliminary definitions We associate to any type σ a one-variable polynomial $P(\sigma)$ defined as $w(\sigma)(X, X, X)$. The depth of σ, written $d(\sigma)$, is by definition the degree of $P(\sigma)$. The size of σ is written $\#\sigma$ and is by definition the size of the finite set $\mathsf{dom}(\sigma)$. The cardinal of the set $\mathsf{ftv}(\sigma)$ is written n_σ.

Properties 2.7.9 *We have the following properties:*
i) *For any polytype* σ, *each coefficient in* $P(\sigma)$ *is bounded by* $\#\sigma - n_\sigma$.
ii) *For any polytype* σ, *we have* $d(\sigma) \leqslant \#\sigma$.

The proof is by structural induction on σ. See details in the Appendix (page 256).

A weak ascending sequence of polytypes $(\sigma_i)_{i\in\mathbb{N}}$ is a sequence such that $(Q)\ \sigma_i \sqsubseteq\!\!\!\!-\ \sigma_{i+1}$ holds for all $i \geqslant 0$. A weak descending sequence is such that $(Q)\ \sigma_{i+1} \sqsubseteq\!\!\!\!-\ \sigma_i$ (or, equivalently, $(Q)\ \sigma_i \sqsupseteq\!\!\!\!-\ \sigma_{i+1}$).

Corollary 2.7.10 *Any weak ascending or weak descending sequence* $(\sigma_i)_{i\in\mathbb{N}}$ *is stationary, that is, there exists* n *such that* $(Q)\ \sigma_i \equiv \sigma_n$ *holds for any* $i \geqslant n$

Proof: By Property 2.1.3.i (page 65), we have $\sigma_i/ = \sigma_{i+1}/$ for any i. Hence, for any i, we have $\sigma_i/ = \sigma_0/$ and $\#\sigma_i = \#\sigma_0$. Let w_i be $w(\sigma_i)$. If the sequence is ascending, we have $(Q)\ \sigma_i \sqsubseteq\!\!\!\!-\ \sigma_{i+1}$, thus $w_i \geqslant w_{i+1}$ holds by Property 2.7.6.i. If the sequence is descending, we have $w_i \leqslant w_{i+1}$ for all i. Let P_i be the polynomial $w_i(X, X, X)$.

- We say that P is bounded by n, when each coefficient in P is bounded by n, and the degree of P is bounded by n too. By Property 2.7.9.i, each coefficient in P_i is bounded by $\#\sigma_i$, that is, by $\#\sigma_0$. The degree of P_i is also bounded by $\#\sigma_0$ by Property 2.7.9.ii. Hence, P_i is bounded by a constant $\#\sigma_0$. We note that P_i is in $\mathbb{N}[X]$ (the coefficients of P cannot be negative).

- The subset S of $\mathbb{N}[X]$ of polynomials P bounded by n is finite, for any n. Hence, the increasing (or decreasing) sequence $(P_i)_{i \in \mathbb{N}}$, which is included in S, is stationary. Consequently, there exists a polynomial P and k such that for any $i \geqslant k$, we have $P_i = P$.
- The set of polynomials w in $\mathbb{N}[X][Y][Z]$ such that $w(X, X, X)$ is P is finite too (indeed, the equation $n_x + n_y + n_z = n$, where n is given, admits a finite number of solutions in $\mathbb{N}^3$.) Hence, the increasing (or decreasing) sequence (w_i) is stationary. Consequently, there exists n such that, by Property 2.7.6.i, $(Q)\ \sigma_i \equiv \sigma_n$ holds for $i \geqslant n$. This is the expected result.

■

In summary, any weak ascending sequence is finite, which also means that the abstraction relation can be considered as a well-founded order (up to equivalence) on the set of types. Such a result will be useful to show confluence of the abstraction relation in Lemma 2.8.2.

2.8 Confluence

We wish to show two important confluence results. The first one is the confluence of the abstraction relation (see Lemma 2.8.2). The second one is the commutation of the abstraction relation and the instance relation, as stated in the Diamond Lemma (Lemma 2.8.4). To begin with, we show some auxiliary confluence results on the atomic relations defined in Section 2.6. The properties below are used only in Lemma 2.8.2 and (indirectly) in Lemma 2.8.4. These lemmas state the confluence of $\mathrel{\sqsubset\!\!\!\!-}$ on the one hand, and of $\mathrel{\sqsubset\!\!\!\!-}$ and $\sqsubseteq$ on the other hand.

Properties 2.8.1

i) *Assume σ_1 is in normal form and Q is unconstrained. If we have $(Q)\ \sigma_1 \mathrel{\dot{\sqsubset}\!\!\!\!-}^{\bar{\alpha}} \sigma_2$ and $(Q)\ \sigma_1 \mathrel{\dot{\sqsubset}\!\!\!\!-}^{\bar{\alpha}} \sigma_3$, then there exists σ_4 such that $(Q)\ \sigma_2 \mathrel{\sqsubset\!\!\!\!-} \sigma_4$ and $(Q)\ \sigma_3 \mathrel{\sqsubset\!\!\!\!-} \sigma_4$.*

ii) *The relation $(\equiv\mathrel{\dot{\sqsubset}\!\!\!\!-}^{\emptyset})$ is weakly confluent under an unconstrained prefix.*

iii) *Assume σ_1 is in normal form and Q is unconstrained. If we have $(Q)\ \sigma_1 \mathrel{\dot{\sqsubset}} \sigma_2$ and $(Q)\ \sigma_1 \mathrel{\dot{\sqsubset}\!\!\!\!-} \sigma_3$, then there exist σ_4 such that $(Q)\ \sigma_2 \mathrel{\dot{\sqsubset}\!\!\!\!-} \sigma_4$ and $(Q)\ \sigma_3 \mathrel{\dot{\sqsubset}} \sigma_4$ holds.*

iv) *Assume $C_f(\sigma_1)$ is in normal form, $\mathsf{level}(C_f) > 1$, and Q is unconstrained. If we have $(Q\overline{C_f})\ \sigma_1 \mathrel{\dot{\sqsubset}\!\!\!\!-}^{\mathsf{dom}(C_f)} \sigma_2$, and $(Q)\ C_f(\sigma_1) \mathrel{\dot{\sqsubset}\!\!\!\!-}^{\bar{\alpha}} \sigma_3$, then there exists σ_4 such that $(Q)\ C_f(\sigma_2) \mathrel{\dot{\sqsubset}\!\!\!\!-}^{\bar{\alpha}} \sigma_4$ and $(Q)\ \sigma_3 \mathrel{\dot{\sqsubseteq}} \sigma_4$ hold.*

These results are shown by considering the critical pairs.
See the full proof in the Appendix (page 257).

The following diagram illustrates the confluence of the abstraction relation (under an unconstrained prefix).

$$\begin{tikzcd}
\sigma_1 \arrow[r, "\mathrel{-\!\!\!\sqsubseteq}"] \arrow[d, "\mathrel{-\!\!\!\sqsubseteq}"'] & \sigma_2 \arrow[d, dotted, "\mathrel{-\!\!\!\sqsubseteq}"] \\
\sigma_3 \arrow[r, dotted, "\mathrel{-\!\!\!\sqsubseteq}"'] & \sigma_4
\end{tikzcd}$$

This result is stated formally by the following lemma.

Lemma 2.8.2 (Confluence of the abstraction relation) *If* $\sigma_1 \mathrel{-\!\!\!\sqsubseteq} \sigma_2$ *and* $\sigma_1 \mathrel{-\!\!\!\sqsubseteq} \sigma_3$ *hold, then there exists* σ_4 *such that* $\sigma_2 \mathrel{-\!\!\!\sqsubseteq} \sigma_4$ *and* $\sigma_3 \mathrel{-\!\!\!\sqsubseteq} \sigma_4$ *hold.*

Proof: By Property 2.6.2.i (page 85), the relations $(\equiv\dot{\mathrel{-\!\!\!\sqsubseteq}}^{\emptyset})^*$ and $\mathrel{-\!\!\!\sqsubseteq}$ are equivalent. Hence, by hypothesis, we can derive both $\sigma_1\ (\equiv\dot{\mathrel{-\!\!\!\sqsubseteq}}^{\emptyset})^*\ \sigma_2$ and $\sigma_1\ (\equiv\dot{\mathrel{-\!\!\!\sqsubseteq}}^{\emptyset})^*\ \sigma_3$. By Property 2.8.1.ii, $(\equiv\dot{\mathrel{-\!\!\!\sqsubseteq}}^{\emptyset})$ is weakly confluent. Besides, by Corollary 2.7.10, $(\equiv\dot{\mathrel{-\!\!\!\sqsubseteq}}^{\emptyset})$ is well-founded (up to equivalence). Note that $(\equiv\dot{\mathrel{-\!\!\!\sqsubseteq}}^{\emptyset})$ is defined up to equivalence. Hence, $(\equiv\dot{\mathrel{-\!\!\!\sqsubseteq}}^{\emptyset})^*$ is confluent, that is, there exists σ_4 such that $\sigma_2\ (\equiv\dot{\mathrel{-\!\!\!\sqsubseteq}}^{\emptyset})^*\ \sigma_4$ and $\sigma_3\ (\equiv\dot{\mathrel{-\!\!\!\sqsubseteq}}^{\emptyset})^*\ \sigma_4$. We can equivalently write $\sigma_2 \mathrel{-\!\!\!\sqsubseteq} \sigma_4$ and $\sigma_3 \mathrel{-\!\!\!\sqsubseteq} \sigma_4$. This is the expected result. ■

The instance relation $\sqsubseteq$ is not confluent. This is not surprising since the ML instance relation, which is a subcase, is not: two incompatible instantiations of the same type variable cannot be merged. However, abstraction and instance commute, as described by the following diagram, to be taken under an unconstrained prefix.

$$\begin{tikzcd}
\sigma_1 \arrow[r, "\sqsubseteq"] \arrow[d, "\mathrel{-\!\!\!\sqsubseteq}"'] & \sigma_2 \arrow[d, dotted, "\mathrel{-\!\!\!\sqsubseteq}"] \\
\sigma_3 \arrow[r, dotted, "\sqsubseteq"'] & \sigma_4
\end{tikzcd}$$

Lemma 2.8.3 *Assume* Q *is unconstrained. If we have* $(Q)\ \sigma_1\ (\equiv\dot{\sqsubseteq})\ \sigma_2$ *and* $(Q)\ \sigma_1\ (\equiv\dot{\mathrel{-\!\!\!\sqsubseteq}}^{\emptyset})\ \sigma_3$*, then there exists* σ_4 *such that* $(Q)\ \sigma_2\ (\equiv\dot{\mathrel{-\!\!\!\sqsubseteq}}^{\emptyset})\ \sigma_4$ *and* $(Q)\ \sigma_3\ (\equiv\dot{\sqsubseteq})\ \sigma_4$ *hold.*

Proof: If $(Q)\ \sigma_1 \equiv \sigma_2$ or $(Q)\ \sigma_1 \equiv \sigma_3$ hold, we get the expected result by taking (respectively) $\sigma_4 = \sigma_3$ or $\sigma_4 = \sigma_2$. Otherwise, we have by definition

$$(Q)\ \sigma_1 \equiv \sigma_1^a\ (\underline{\mathbf{1}}) \qquad (Q)\ \sigma_1^a \mathrel{\dot{\sqsubseteq}} \sigma_2' \qquad (Q)\ \sigma_2' \equiv \sigma_2 \qquad (Q)\ \sigma_1 \equiv \sigma_1^b \qquad (Q)\ \sigma_1^b\ \dot{\mathrel{-\!\!\!\sqsubseteq}}^{\bar{\alpha}}\ \sigma_3'$$

$$(Q)\ \sigma_3' \equiv \sigma_3$$

Let σ'_1 be $\mathsf{nf}(\sigma_1)$. By Property 1.5.6.iv (page 51), σ'_1 is in normal form and by Property 1.5.6.i (page 51), $(Q)\ \sigma'_1 \equiv \sigma_1$ holds. Besides, by Property 1.5.11.i (page 54) and (1), $\sigma'_1 \approx \mathsf{nf}(\sigma_1^a)$ holds. By Property 2.6.3.i (page 86), there exists σ''_2 such that $(Q)\ \sigma''_2 \equiv \sigma'_2$ and $(Q)\ \sigma'_1\ \dot{\sqsubseteq}\ \sigma''_2$. Similarly, there exists σ''_3 such that $(Q)\ \sigma''_3 \equiv \sigma'_3$ and $(Q)\ \sigma'_1\ \dot{⋿}^{\bar\alpha}\ \sigma''_3$. Hence, we have

$$(Q)\ \sigma_1 \equiv \sigma'_1 \qquad (Q)\ \sigma'_1\ \dot{\sqsubseteq}\ \sigma''_2\ (\underline{\mathbf{2}}) \qquad (Q)\ \sigma'_1\ \dot{⋿}^{\bar\alpha}\ \sigma''_3 \qquad (Q)\ \sigma''_2 \equiv \sigma_2 \qquad (Q)\ \sigma''_3 \equiv \sigma_3$$

We have three rules to derive (2):

∘ CASE C-STRICT: Then $(Q)\ \sigma'_1\ \dot{\sqsubset}\ \sigma''_2$ holds. By Property 2.8.1.iii, there exists σ_4 such that $(Q)\ \sigma''_3\ \dot{\sqsubset}\ \sigma_4$ and $(Q)\ \sigma''_2\ \dot{⋿}^{\bar\alpha}\ \sigma_4$. Hence, $(Q)\ \sigma_3\ (\equiv\dot{\sqsubseteq})\ \sigma_4$ and $(Q)\ \sigma_2\ (\equiv\dot{⋿}^{\emptyset})\ \sigma_4$ hold, which is the expected result.

∘ CASE C-ABSTRACT-F: Then by Property 2.8.1.iv, there exists σ_4 such that $(Q)\ \sigma''_2\ \dot{⋿}^{\bar\alpha}\ \sigma_4$ and $(Q)\ \sigma''_3\ \dot{\sqsubseteq}\ \sigma_4$. Hence, we get $(Q)\ \sigma_2\ (\equiv\dot{⋿}^{\emptyset})\ \sigma_4$ and $(Q)\ \sigma_3\ (\equiv\dot{\sqsubseteq})\ \sigma_4$, which is the expected result.

∘ CASE C-ABSTRACT-R: then, $(Q)\ \sigma_1 ⋿ \sigma_2$ and $(Q)\ \sigma_1 ⋿ \sigma_3$, thus we conclude directly by Lemma 2.8.2. ■

Lemma 2.8.4 (Diamond Lemma) *If Q is unconstrained, $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ and $(Q)\ \sigma_1 ⋿ \sigma_3$ hold, then there exist σ_4 such that $(Q)\ \sigma_2 ⋿ \sigma_4$ and $(Q)\ \sigma_3 \sqsubseteq \sigma_4$ hold.*

Proof: By Property 2.6.2.i (page 85), the relation $⋿$ is equivalent to $(\equiv\dot{⋿}^{\emptyset})^*$ and the relation $\sqsubseteq$ is equivalent to $(\equiv\dot{\sqsubseteq})^*$. Lemma 2.8.3 states the strong confluence of $(\equiv\dot{⋿}^{\emptyset})$ and $(\equiv\dot{\sqsubseteq})$, hence, there exist σ_4 such that $(Q)\ \sigma_2\ (\equiv\dot{⋿}^{\emptyset})^*\ \sigma_4$ and $(Q)\ \sigma_3\ (\equiv\dot{\sqsubseteq})\ \sigma_4$. By Properties 2.6.2.i (page 85) and 2.6.2.ii (page 85), this amounts to writing $(Q)\ \sigma_2 ⋿ \sigma_4$ and $(Q)\ \sigma_3 \sqsubseteq \sigma_4$. ■

The diamond lemma is stated under an unconstrained prefix. This restriction is mandatory, as shown by the following counter-example. Take

$$\sigma_1 \triangleq \forall\,(\alpha)\ \alpha \to \alpha \qquad Q \triangleq (\beta = \sigma_1) \qquad \sigma_2 \triangleq \mathtt{int} \to \mathtt{int} \qquad \sigma_3 \triangleq \alpha$$

Graphically, this gives

$$\begin{tikzcd}
\forall\,(\alpha)\ \alpha \to \alpha \arrow[r, "\sqsubseteq"] \arrow[d, "⋿"'] & \mathtt{int} \to \mathtt{int} \arrow[d, dotted, "⋿"] \\
\alpha \arrow[r, dotted, "\sqsubseteq"'] & ?
\end{tikzcd}$$

It is impossible to find any suitable σ_4 which closes the diagram. Indeed, the only instance of α are types equivalent to α, which cannot be abstractions of $\mathtt{int} \to \mathtt{int}$.

Chapter 3

Relations between prefixes

Prefixes play a crucial role in ML^{F}. Whereas the solutions to a unification problem are substitutions in ML, that is, mappings from type variables to monotypes, in ML^{F} the solutions to a unification problem are prefixes, that is, mappings from type variables to types (possibly polytypes). Hence, prefixes can be seen as a generalization of the notion of substitutions to polytypes.

In Section 3.1, we show how a prefix can express a monotype substitution, just as in ML. A substitution θ_1 is said more general than a substitution θ_2 when there exists θ such that $\theta_2 = \theta \circ \theta_1$. Conversely, we can say that θ_2 is an instance of θ_1. We expect a similar instance relation on prefixes in ML^{F}. More precisely, rules R-Context-Rigid and R-Context-Flexible show that two types $\forall\,(Q)\;\sigma$ and $\forall\,(Q')\;\sigma$ with the same suffix can be in an instance relation, for any suffix σ. This suggests a notion of inequality between prefixes alone. However, because prefixes are "open" this relation must be defined relative to a set of variables that lists (a superset of) the free type variables of σ, called an *interface* and written with letter I. Prefix relations are defined in Section 3.2. Then we state a few results about prefixes and prefix instance. The main result of this chapter is Lemma 3.6.4, which implies that if $(Q)\;\sigma_1 \sqsubseteq \sigma_2$ $\mathbf{(\underline{1})}$ holds and Q' is an instance of Q, then $(Q')\;\sigma_1 \sqsubseteq \sigma_2$ $\mathbf{(\underline{2})}$ also holds. Such a result is used for showing the soundness of the unification algorithm. Indeed, it shows that if an instance relation (1) holds under a prefix Q, then it also holds (2) under the instantiation of its prefix.

3.1 Substitutions

A *monotype prefix* is a prefix whose bounds are all in $\mathcal{T}$. For example $(\alpha_1 = \tau_1, \ldots \alpha_n = \tau_n)$ is a monotype prefix, which embeds the substitution $[\tau_1/\alpha_1] \circ \ldots \circ [\tau_n/\alpha_n]$ within type expressions[1]. Conversely, given an idempotent substitution θ, we write $\underline{\theta}$ for the corresponding monotype prefix. More precisely, $\underline{\theta}$ is the prefix $(\alpha = \theta(\alpha))_{\alpha \in \mathsf{dom}(\theta)}$. Note that since θ is idempotent, $\underline{\theta}$ is automatically a well-formed prefix. As expected, we have $(Q)\ \forall\,(\underline{\theta})\ \sigma \equiv \theta(\sigma)$ and $(Q\underline{\theta})\ \sigma \equiv \theta(\sigma)$ (the former by Rule Eq-Mono* and the latter by iteration of Rule Eq-Mono). The following rule is derivable using R-Context-R and Eq-Mono*:

Pr-Subst*

$$\frac{(Q\underline{\theta})\ \sigma_1 \Diamond \sigma_2}{(Q)\ \theta(\sigma_1) \Diamond \theta(\sigma_2)}$$

Note also that the substitution extracted from $\underline{\theta}$, that is $\widehat{\underline{\theta}}$, is equal to θ.

In Section 1.3.4 we defined the application of a substitution θ to a prefix Q. In the particular case where Q is itself a substitution $\underline{\theta'}$, we may wonder what is $\theta(Q)$. The answer is expressed in Property 3.1.1.i below. The case where θ is a renaming ϕ corresponds to Property 3.1.1.ii. Finally, Property 3.1.1.iii gives the substitution extracted from $\theta(Q)$.

Properties 3.1.1 *If we have* $\mathsf{dom}(\theta') \# \mathsf{dom}(\theta) \cup \mathsf{codom}(\theta)$*, then the following properties hold:*

i) *The prefix* $\theta(\underline{\theta'})$ *is a monotype prefix corresponding to the substitution* $\theta \circ \theta'$ *restricted to* $\mathsf{dom}(\theta')$*.*

ii) *The prefix* $\phi(\underline{\theta'})$ *is a monotype prefix corresponding to the substitution* $\phi \circ \theta' \circ \phi^{\neg}$*.*

iii) *For any prefix* Q *and renaming* ϕ *on* $\mathsf{dom}(Q)$*, we have* $\widehat{\phi(Q)} = \phi \circ \widehat{Q} \circ \phi^{\neg}$*.*

Proof: Property i: By construction, $\theta(\underline{\theta'})$ is a monotype prefix. The domain of $\theta(\underline{\theta'})$ is by definition $\mathsf{dom}(\underline{\theta'})$, that is, $\mathsf{dom}(\theta')$. Let α be in $\mathsf{dom}(\theta')$. By definition, $\theta(\underline{\theta'})(\alpha)$ is $\theta(\theta'(\alpha))$, that is, $\theta \circ \theta'(\alpha)$. Hence, $\theta(\underline{\theta'})$ and $\theta \circ \theta'$ are equal on $\mathsf{dom}(\theta')$.

Property ii: By definition, $\phi(\underline{\theta'})$ is a monotype prefix whose associated substitution θ'' is defined as follows: $\alpha \in \mathsf{dom}(\theta'')$ if and only if there exists $\beta \in \mathsf{dom}(\theta')$ such that $\phi(\beta) = \alpha$; then $\theta''(\alpha)$ is $\phi(\theta'(\beta))$ $(\underline{\mathbf{1}})$. Given any α, we have $\phi \circ \theta' \circ \phi^{\neg}(\alpha) = \phi \circ \theta'(\beta)$, where α is $\phi(\beta)$. If $\alpha \in \mathsf{dom}(\theta'')$, then $\phi \circ \theta'(\beta) = \theta''(\alpha)$ by (1). Otherwise, $\theta''(\alpha)$ is α and $\beta \notin \mathsf{dom}(\theta')$. Hence, $\theta'(\beta) = \beta$, thus $\phi \circ \theta'(\beta) = \phi(\beta) = \alpha$. In both cases, we have $\phi \circ \theta' \circ \phi^{\neg}(\alpha) = \theta''(\alpha)$. This holds for all α, therefore θ'' is $\phi \circ \theta' \circ \phi^{\neg}$.

Property iii: By definition, $\widehat{\phi(Q)}$ is $\phi(\widehat{Q})$. We get the expected result by Property ii. ■

---

[1] Actually, prefixes are slightly more precise than substitutions because they are kept as a sequence of elementary substitutions rather than just their composition.

3.2 Prefix instance

As explained above, we define an instance relation on prefixes, relative to a set of type variables I, called an interface. For example, we expect the prefixes $(\beta \geq \bot, \alpha = \beta \to \beta)$ and $(\gamma \geq \bot, \alpha = \gamma \to \gamma)$ to be equivalent under the interface $\{\alpha\}$, but not under the interface $\{\alpha, \beta\}$ or $\{\alpha, \gamma\}$. To ease the presentation, we introduce a new notation: We write Σ_I for the set of types whose unbound variables are in I.

Definition 3.2.1 (Prefix instance) Let Q and Q' be closed well-formed prefixes, and I be a set of variables such that $I \subseteq \mathsf{dom}(Q) \cap \mathsf{dom}(Q')$ holds. A prefix Q' is an *instance* of a prefix Q under the interface I, and we write $Q \sqsubseteq^I Q'$, if and only if $\forall\,(Q)\ \sigma \sqsubseteq \forall\,(Q')\ \sigma$ holds for all types σ in Σ_I. We omit I in the notation when it is equal to $\mathsf{dom}(Q)$. We define $Q \equiv^I Q'$ and $Q \sqsubseteq\!\!\!-^I Q'$ similarly. □

Let Q be a well-formed closed prefix. Then $\emptyset \equiv Q$ holds, which means by notation $\emptyset \equiv^\emptyset Q$. Conversely, $Q \equiv^I \emptyset$ holds if and only if I is empty. In particular, $Q \equiv \emptyset$ does not hold if Q is not empty. Although the relation $\equiv^I$ is symmetric, we see that the relation $\equiv$ (where the interface is left implicit) is not.

Example 3.2.8 For any σ, we can derive $(\emptyset)\ \forall\,(\alpha \geq \bot)\ \sigma \sqsubseteq \forall\,(\beta)\ \forall\,(\alpha \geq \beta \to \beta)\ \sigma$ by Eq-Free, I-Nil, and context rules. More generally, if we have Q and Q' unconstrained, $\mathsf{dom}(\theta) \subseteq \mathsf{dom}(Q)$, and $\mathsf{codom}(\theta) \subseteq \mathsf{dom}(Q')$, then we have $Q \sqsubseteq Q'\underline{\theta}$. This covers the instantiation of substitutions in ML, where free variables have the implicit bound $\bot$, just as in Q (which is unconstrained), and where free variables can be substituted by any monotype.

Properties 3.2.2

i) *We have* $Q_1 \equiv^I Q_2$ *iff* $Q_1 \sqsubseteq^I Q_2$ *and* $Q_2 \sqsubseteq^I Q_1$.

ii) *If* $Q_1 \approx Q_2$*, then* $Q_1 \equiv Q_2$.

It is a direct consequence of Property 2.7.7.i (page 96). See Appendix (page 265).

Assume we have $(Q)\ \sigma \sqsubseteq \alpha$ and σ is not a type variable. Then σ is instantiated to a type variable bound in the prefix. The only way to do so is by either Rule A-Hyp or Rule I-Hyp. This means that σ can be instantiated into the bound of α. This is expressed formally by the following properties.

Properties 3.2.3

i) *If we have* $(Q)\ \sigma \sqsubseteq\!\!\!- \alpha$ *and* $\sigma \notin \mathcal{V}$*, then* $(Q)\ \sigma \sqsubseteq\!\!\!- Q(\alpha)$ *holds.*

ii) *If we have* $(Q)\ \sigma \sqsubseteq \alpha$ *and* $\sigma \notin \mathcal{V}$*, then* $(Q)\ \sigma \sqsubseteq Q(\alpha)$ *holds.*

iii) *For all* Q *and* α*,* $(Q)\ Q(\alpha) \sqsubseteq \alpha$ *holds.*

See proof in the Appendix (page 266).

3.3 Domains of prefixes

As seen in Section 1.2, if Q is the prefix $(\alpha_1 \diamond_1 \sigma_1, \ldots \alpha_n \diamond_n \sigma_n)$, then its domain $\mathsf{dom}(Q)$ is the set $\{\alpha_1, \ldots \alpha_n\}$. We need to capture the notion of useful domain of a prefix Q.

Definition 3.3.1 If I is a set of type variables, the *domain of Q useful for I*, written $\mathsf{dom}(Q/I)$, is defined as follows:

$$\alpha \in \mathsf{dom}(Q/I) \text{ if and only if } Q = (Q_1, \alpha \diamond \sigma, Q_2) \text{ and } \alpha \in \mathsf{ftv}(\forall\,(Q_2)\ \nabla_I)$$

We also write $\mathsf{dom}(Q/\sigma)$ instead of $\mathsf{dom}(Q/\,\mathsf{ftv}(\sigma))$ and similarly $\mathsf{dom}(Q/\sigma_1, \sigma_2)$ instead of $\mathsf{dom}(Q/\,\mathsf{ftv}(\sigma_1) \cup \mathsf{ftv}(\sigma_2))$. □

Intuitively, $\mathsf{dom}(Q/\sigma)$ is the domain of Q which is useful for σ. For example, if Q' corresponds to Q, where all bindings not in $\mathsf{dom}(Q/\sigma)$ have been removed, then we have $\forall\,(Q)\ \sigma \equiv \forall\,(Q')\ \sigma$ by Eq-Free.

Properties 3.3.2
i) *We have* $\mathsf{dom}(Q/I \cup J) = \mathsf{dom}(Q/I) \cup \mathsf{dom}(Q/J)$.
ii) *If* $Q \approx Q'$ *holds, and* $\bar\alpha \subseteq \mathsf{dom}(Q)$*, then* $\mathsf{dom}(Q/\bar\alpha) = \mathsf{dom}(Q'/\bar\alpha)$.

See proof in the Appendix (page 268).

The notion of useful domain of a prefix can be viewed as an extension of the notion of free variables. Indeed, in ML, all free variables have the bound $\bot$, which means that the implicit prefix Q is unconstrained. Then the domain of Q useful for a polytype σ, *i.e.* $\mathsf{dom}(Q/\sigma)$, is exactly the set of free variables of σ. In ML$^{\mathrm{F}}$, however, a variable β can be in $\mathsf{dom}(Q/\sigma)$ directly or indirectly. For a direct example, take $\forall\,(\alpha \geq \bot)\ \alpha \to \beta$ for σ, and $(\beta \geq \bot)$ for Q. For an indirect example, take $\forall\,(\alpha \geq \bot)\ \alpha \to \gamma$ for σ, and $(\beta \geq \bot, \gamma \geq \forall\,(\delta)\ \delta \to \beta)$ for Q. In the indirect example, β is not free in σ, but γ is free in σ and β is free in the bound of γ. We see that β is "indirectly" free in σ via the prefix Q. This is captured by the notion of useful domain of the prefix, which is therefore an extension of the notion of free variables.

3.4 Rules for prefix equivalence, abstraction, and instance

Prefix instance was defined in Definition 3.2.1. Below, we give a syntactic characterization. Inference rules defining prefix equivalence, prefix abstraction, and prefix instance are given in figures 3.1, 3.2, and 3.3. As in Section 1.7, we use the symbol $\diamondsuit_\ell$ as a meta-variable standing for $\equiv_\ell$, $\sqsubseteq_\ell$, or $\sqsubseteq_\ell$. The notation $Q \diamondsuit_\ell^I Q'$ implicitly requires Q and Q' to be closed well-formed prefixes such that $I \subseteq \mathsf{dom}(Q) \cap \mathsf{dom}(Q')$.

Figure 3.1: Prefix equivalence

PE-Refl

$$Q \equiv^I_\ell Q$$

PE-Trans

$$\frac{Q_1 \equiv^I_\ell Q_2 \qquad Q_2 \equiv^I_\ell Q_3}{Q_1 \equiv^I_\ell Q_3}$$

PE-Free

$$\frac{\alpha \notin \mathsf{dom}(Q) \qquad \alpha \notin I}{Q \equiv^I_\ell (Q, \alpha \diamond \sigma)}$$

PE-Mono

$$\frac{}{(Q, \alpha \geq \tau, Q_0) \equiv^I_\ell (Q, \alpha = \tau, Q_0)}$$

PE-Context-L

$$\frac{(Q)\ \sigma_1 \equiv \sigma_2}{(Q, \alpha \diamond \sigma_1, Q_0) \equiv^I_\ell (Q, \alpha \diamond \sigma_2, Q_0)}$$

PE-Swap

$$\frac{\sigma \notin \mathcal{T}}{(Q, \alpha_1 \diamond \sigma, \alpha_2 = \alpha_1, Q_0) \equiv^I_\ell (Q, \alpha_2 \diamond \sigma, \alpha_1 = \alpha_2, Q_0)}$$

PE-Comm

$$\frac{\alpha_1 \notin \mathsf{ftv}(\sigma_2) \qquad \alpha_2 \notin \mathsf{ftv}(\sigma_1)}{(Q, \alpha_1 \diamond_1 \sigma_1, \alpha_2 \diamond_2 \sigma_2, Q_0) \equiv^I_\ell (Q, \alpha_2 \diamond_2 \sigma_2, \alpha_1 \diamond_1 \sigma_1, Q_0)}$$

Figure 3.2: Prefix Abstraction

PA-Equiv

$$\frac{Q_1 \equiv^I_\ell Q_2}{Q_1 \sqsubseteq\!\!\!\!\!-^I_\ell Q_2}$$

PA-Trans

$$\frac{Q_1 \sqsubseteq\!\!\!\!\!-^I_\ell Q_2 \qquad Q_2 \sqsubseteq\!\!\!\!\!-^I_\ell Q_3}{Q_1 \sqsubseteq\!\!\!\!\!-^I_\ell Q_3}$$

PA-Context-L

$$\frac{(Q)\ \sigma_1 \sqsubseteq\!\!\!\!\!- \sigma_2}{(Q, \alpha = \sigma_1, Q_0) \sqsubseteq\!\!\!\!\!-^I_\ell (Q, \alpha = \sigma_2, Q_0)}$$

Unsurprisingly, these rules are similar to the rules defining equivalence, abstraction, and instantiation of types, except for Rule PE-Swap, which corresponds to (implicit) α-conversion between types.

We wish to show the equivalence between the relation $\sqsubseteq^I_\ell$ and the relation $\sqsubseteq^I$. To begin with, we check that $\sqsubseteq^I_\ell$ is included in $\sqsubseteq^I$.

Lemma 3.4.1 *If $Q \Diamond^I_\ell Q'$ holds, then $Q \Diamond^I Q'$.*

Proof: We show by induction on the derivation of $Q \Diamond^I_\ell Q'$ that for any σ such that $\mathsf{ftv}(\sigma) \subseteq I$, we have $\forall (Q)\ \sigma \Diamond \forall (Q')\ \sigma$. All cases are easy. ■

Figure 3.3: Prefix instance

$$\frac{Q_1 \mathrel{-\!\!\!\!\sqsubseteq}^I_\ell Q_2}{Q_1 \sqsubseteq^I_\ell Q_2}\ \text{(PI-Abstract)} \qquad \frac{Q_1 \sqsubseteq^I_\ell Q_2 \quad Q_2 \sqsubseteq^I_\ell Q_3}{Q_1 \sqsubseteq^I_\ell Q_3}\ \text{(PI-Trans)} \qquad \frac{}{(Q, \alpha \geq \sigma_1, Q_0) \sqsubseteq^I_\ell (Q, \alpha = \sigma_1, Q_0)}\ \text{(PI-Rigid)}$$

$$\frac{(Q)\ \sigma_1 \sqsubseteq \sigma_2}{(Q, \alpha \geq \sigma_1, Q_0) \sqsubseteq^I_\ell (Q, \alpha \geq \sigma_2, Q_0)}\ \text{(PI-Context-L)}$$

The following rule is derivable by PE-Context-L and Eq-Mono:

$$\frac{(Q)\ \sigma \equiv \tau}{(Q, \alpha = \sigma, Q') \equiv_\ell (Q, \alpha = \sigma, Q'[\tau/\alpha])}\ \text{(PE-Mono}^\star\text{)}$$

We now prove some useful properties about prefix relations. For instance, Property i below shows that the interface of a relation can always be shrinked. Notice also that $Q_1 \diamondsuit^\emptyset_\ell Q_2$ always holds, for any prefixes Q_1 and Q_2. Property ii simply states that derivations are stable under renaming. Property iii shows that it is always possible to append a non-interfering prefix Q to both sides of a derivation of $Q_1 \diamondsuit^I_\ell Q_2$. Property iv shows that the equivalence between a prefix a renaming of this prefix can be derived. Property v states if a sequence of binders is duplicated in a prefix (up to renaming), both sequences can be merged by instantiation.

Properties 3.4.2

i) *If* $Q_1 \diamondsuit^I_\ell Q_2$ *and* $J \subseteq I$ *hold, then we have* $Q_1 \diamondsuit^J_\ell Q_2$.

ii) *If* ϕ *is a renaming of* $\mathsf{dom}(Q) \cup \mathsf{dom}(Q')$, *and if there is a derivation of* $Q \diamondsuit^I_\ell Q'$ *of size* n, *then we have a derivation of* $\phi(Q) \diamondsuit^{\phi(I)}_\ell \phi(Q')$ *of size* n.

iii) *If* $Q_1 \diamondsuit^I_\ell Q_2$, $Q \# Q_1$, $Q \# Q_2$, *and* $\mathsf{utv}(Q) \subseteq I$, *then* $Q_1 Q \diamondsuit^{I \cup \mathsf{dom}(Q)}_\ell Q_2 Q$.

iv) *If* ϕ *is a renaming of* $\mathsf{dom}(Q)$ *and* Q *a closed well-formed prefix, then we have* $Q \equiv_\ell \phi(Q)\underline{\phi}$.

v) *If* ϕ *is a renaming of domain* $\mathsf{dom}(Q)$ *such that* $Q_1 Q \phi(Q) Q_2$ *is a well-formed closed prefix, then we can derive* $Q_1 Q \phi(Q) Q_2 \sqsubseteq_\ell Q_1 Q \underline{\phi^\neg} Q_2$.

See proof in the Appendix (page 268).

As explained in the introduction of this chapter, the main result is Lemma 3.6.4 (page 114). A less general case is stated by the next lemma. Namely, if two types σ

and σ' are equivalent under a prefix Q, and if Q' is an instance of Q, then we expect σ and σ' to be equivalent under Q' too. This statement is refined by providing the interface I of the instance relation, as well as the set of free variables of σ and σ'. We recall that Σ_I is the set of types whose unbound variables are in I.

Lemma 3.4.3 *If we have $Q_1 \diamondsuit_\ell^I Q_2$, then for all σ and σ' in Σ_I such that $(Q_1)\ \sigma \equiv \sigma'$ holds, we have $(Q_2)\ \sigma \equiv \sigma'$.*

Proof: By hypothesis, we have $Q_1 \diamondsuit_\ell^I Q_2$ $\mathbf{(\underline{1})}$, $(Q_1)\ \sigma \equiv \sigma'$ $\mathbf{(\underline{2})}$, and $\mathsf{ftv}(\sigma) \cup \mathsf{ftv}(\sigma') \subseteq I$ $\mathbf{(\underline{3})}$. We have to show that $(Q_2)\ \sigma \equiv \sigma'$ $\mathbf{(\underline{4})}$ holds. We note that it suffices to show that $\widehat{Q_1}$ and $\widehat{Q_2}$ are equal on I $\mathbf{(\underline{5})}$. Indeed, Corollary 1.5.10 and (2) imply $\widehat{Q_1}(\sigma) \equiv \widehat{Q_1}(\sigma')$ $\mathbf{(\underline{6})}$. Additionally, (5), and (3) give $\widehat{Q_1}(\sigma) = \widehat{Q_2}(\sigma)$ $\mathbf{(\underline{7})}$ and $\widehat{Q_1}(\sigma') = \widehat{Q_2}(\sigma')$ $\mathbf{(\underline{8})}$. By (6), (7), and (8), we get $\widehat{Q_2}(\sigma) \equiv \widehat{Q_2}(\sigma')$, which implies (4) thanks to Corollary 1.5.10. We prove either (4) or (5) according to which one is easier. The proof is by induction on the derivation of (1).

∘ Case PE-Refl: Immediate.

∘ Case PE-Trans, PA-Trans, PI-Trans, PA-Equiv, and PI-Abstract: By induction hypothesis.

∘ Case PE-Comm and PI-Rigid: $\widehat{Q_1}$ is $\widehat{Q_2}$, thus (5) holds.

∘ Case PE-Free: We have $Q_2 = (Q_1, \alpha \diamond \sigma)$. Hence, $\widehat{Q_2} = \widehat{Q_1} \circ \theta$, where $\mathsf{dom}(\theta) \subseteq \{\alpha\}$. By hypothesis, $\alpha \notin I$. Hence, $\widehat{Q_2}$ and $\widehat{Q_1}$ are equal on I, and we have shown (5).

∘ Case PE-Mono: By definition, $\widehat{Q_1}$ and $\widehat{Q_2}$ are equal, thus we have (5).

∘ Case PE-Context-L, PA-Context-L and PI-Context-L: By hypothesis, we have $Q_1 = (Q_a, \alpha \diamond \sigma_1, Q_b)$ and $Q_2 = (Q_a, \alpha \diamond \sigma_2, Q_b)$. Besides, $(Q_a)\ \sigma_1 \diamondsuit \sigma_2$ $\mathbf{(\underline{9})}$ holds. Let θ_a be $\widehat{Q_a}$ and θ_b be $\widehat{Q_b}$. If $\alpha \notin \mathsf{dom}(\widehat{Q_1})$, then, $\widehat{Q_1}$ is $\theta_a \circ \theta_b$, and $\widehat{Q_2}$ is $\theta_a \circ \theta \circ \theta_b$, where $\mathsf{dom}(\theta) \subseteq \{\alpha\}$. Hence, $\widehat{Q_2}$ is $\theta_a \circ \theta \circ \theta_a \circ \theta_b$, that is, $\theta_a \circ \theta \circ \widehat{Q_1}$ $\mathbf{(\underline{10})}$. By (6), (10), and Property 1.5.11.v, we get $\theta_a \circ \theta \circ \widehat{Q_1}(\sigma) \equiv \theta_a \circ \theta \circ \widehat{Q_1}(\sigma')$, that is, $\widehat{Q_2}(\sigma) \equiv \widehat{Q_2}(\sigma')$ by (10). This implies (4) by Corollary 1.5.10. Otherwise, $\alpha \in \mathsf{dom}(\widehat{Q_1})$, which means that $\sigma_1 \in \mathcal{T}$. By Lemma 2.1.6 and (9), we get $\sigma_2 \in \mathcal{T}$ and $(Q_a)\ \sigma_1 \equiv \sigma_2$. By Lemmas 1.5.9 and 1.5.6.iii (page 51), we get $\theta_a \circ \widehat{(\alpha \diamond \sigma_1)} = \theta_a \circ \widehat{(\alpha \diamond \sigma_2)}$, thus $\widehat{Q_1} = \widehat{Q_2}$ holds, that is (5).

∘ Case PE-Swap: We have $Q_1 = (Q_a, \alpha_1 \diamond \sigma, \alpha_2 = \alpha_1, Q_b)$ $\mathbf{(\underline{11})}$ and $Q_2 = (Q_a, \alpha_2 \diamond \sigma, \alpha_1 = \alpha_2, Q_b)$. Let θ_a be $\widehat{Q_a}$ and θ_b be $\widehat{Q_b}$. Let θ_1 be $\widehat{(\alpha_1 \diamond \sigma)}$ and θ_2 be $\widehat{(\alpha_2 \diamond \sigma)}$. Note that the substitution $[\alpha_2/\alpha_1] \circ \theta_1 \circ [\alpha_1/\alpha_2]$ is equal to the substitution $\theta_2 \circ [\alpha_2/\alpha_1]$ $\mathbf{(\underline{12})}$ (consider the images of α_1 and α_2 for the two cases $\sigma \in \mathcal{T}$ and $\sigma \notin \mathcal{T}$). Then using these notations, $\widehat{Q_1}$ is $\theta_a \circ \theta_1 \circ [\alpha_1/\alpha_2] \circ \theta_b$ $\mathbf{(\underline{13})}$ and $\widehat{Q_2}$ is $\theta_a \circ \theta_2 \circ [\alpha_2/\alpha_1] \circ \theta_b$ $\mathbf{(\underline{14})}$. By (6) and (13), we have $\theta_a \circ \theta_1 \circ [\alpha_1/\alpha_2] \circ \theta_b(\sigma) \equiv \theta_a \circ \theta_1 \circ [\alpha_1/\alpha_2] \circ \theta_b(\sigma')$. Composing by $[\alpha_2/\alpha_1]$ (Property 1.5.11.v (page 54)), we get $[\alpha_2/\alpha_1] \circ \theta_a \circ \theta_1 \circ [\alpha_1/\alpha_2] \circ \theta_b(\sigma) \equiv [\alpha_2/\alpha_1] \circ \theta_a \circ \theta_1 \circ [\alpha_1/\alpha_2] \circ \theta_b(\sigma')$ $\mathbf{(\underline{15})}$. By well-formedness of (11), we have $\alpha_1 \notin \mathsf{dom}(\theta_a)$ and $\alpha_2 \notin \mathsf{codom}(\theta_a)$. Hence, θ_a and $[\alpha_2/\alpha_1]$ commute: from (15), we get

$\theta_a \circ [\alpha_2/\alpha_1] \circ \theta_1 \circ [\alpha_1/\alpha_2] \circ \theta_b(\sigma) \equiv \theta_a \circ [\alpha_2/\alpha_1] \circ \theta_1 \circ [\alpha_1/\alpha_2] \circ \theta_b(\sigma')$. By (12), we get $\theta_a \circ \theta_2 \circ [\alpha_2/\alpha_1] \circ \theta_b(\sigma) \equiv \theta_a \circ \theta_2 \circ [\alpha_2/\alpha_1] \circ \theta_b(\sigma')$. By (14), this is $\widehat{Q_2}(\sigma) \equiv \widehat{Q_2}(\sigma')$. This implies (4) by Corollary 1.5.10. ■

The instance relation on types $\sqsubseteq$ and the instance relation on prefixes are closely related. In particular, if $Q_1 \sqsubseteq^I Q_2$ holds, and $\tau \in \Sigma_I$, we have by definition $\forall\,(Q_1)\ \tau \sqsubseteq \forall\,(Q_2)\ \tau$ **(<u>1</u>)**. Conversely, if (1) holds, we expect the prefix Q_2 to be an instance of the prefix Q_1, under the interface $\mathsf{ftv}(\tau)$, that is $Q_1 \sqsubseteq^{\mathsf{ftv}(\tau)} Q_2$. Actually, we show in the next lemma that $Q_1 \sqsubseteq_\ell^{\mathsf{ftv}(\tau)} Q_2$ **(<u>2</u>)** holds, which implies $Q_1 \sqsubseteq^{\mathsf{ftv}(\tau)} Q_2$ **(<u>3</u>)** by Lemma 3.4.1. Showing (2) instead of (3) is more general, and is a key result used to show the equality between $\sqsubseteq_\ell^I$ and $\sqsubseteq^I$. In summary, we show in Lemma 3.4.4 that (1) implies (2). The statement of the lemma is, however, a bit more involved.

First, the main hypothesis $(Q)\ \sigma_1 \diamondsuit \sigma_2$ is more general than (1). Additionally, the constructed form of σ_1 is $\forall\,(Q_1)\ \tau_1$, which discards the case where $\mathsf{nf}(\sigma_1)$ is $\bot$. Similarly, the constructed form of σ_2 is $\forall\,(Q_2)\ \tau_2$.

The main conclusion is $QQ_1 \diamondsuit_\ell^{\mathsf{dom}(Q)\cup I} QQ_2\underline{\theta}$ **(<u>4</u>)**, which corresponds to (2) after adding the prefix Q and a substitution θ. The role of the substitution θ is to map variables of Q_1 to variables of Q_2. Indeed, by renaming, $\forall\,(Q_1)\ \tau_1$ and $\forall\,(Q_2)\ \tau_2$ might use different variable names; we use the substitution θ to map names from $\forall\,(Q_1)\ \tau_1$ to names of $\forall\,(Q_2)\ \tau_2$. The correctness of this mapping could be intuitively expressed by the equality $\theta(\tau_1) = \tau_2$ **(<u>5</u>)**. Actually, (5) does not always hold, because the prefixes Q and Q_2 must be taken into account. Lemma 3.4.4 is more precise and provides the correct result, that is, $(QQ_2)\ \theta(\tau_1) \equiv \tau_2$ **(<u>6</u>)**. Note, though, that (5) implies (6). What is more, θ is a substitution, but is not necessarily a renaming. For example, we have $(\alpha)\ \forall\,(\beta = \alpha \to \alpha)\ \beta \to \beta \equiv (\alpha \to \alpha) \to (\alpha \to \alpha)$ by Eq-Mono. In such a case, the left-hand prefix is $(\beta = \alpha \to \alpha)$, and the right-hand prefix is empty. Then θ is the substitution $[\alpha \to \alpha/\beta]$; it is not a renaming.

The lemma considers the constructed forms of σ_1 and σ_2 (see page 55), in order to avoid degenerate cases. A type such as $\sigma_1 \triangleq \forall\,(\alpha = \forall\,(Q'_1)\ \tau_1)\ \alpha$ hides its structure in the bound of α. By Eq-Var, we have $\sigma_1 \equiv \forall\,(Q'_1)\ \tau_1$ **(<u>7</u>)**. As mentioned above, the substitution θ is used to link variables of $\mathsf{dom}(Q_2)$ to variables of $\mathsf{dom}(Q_1)$. If we do not take the constructed form of σ_1, Q_1 would be a single binding $(\alpha = \forall\,(Q'_1)\ \tau_1)$ **(<u>8</u>)**. However, because of the equivalence (7), the meaningful bindings are those of Q'_1, but not only (8). In other words, it is not possible to map variables of $\mathsf{dom}(Q_2)$ to α only, but it is possible to map them to $\mathsf{dom}(Q'_1)$. Taking the constructed form ensures that the considered prefix is always the "meaningful" prefix.

Moreover, Lemma 3.4.4 assumes that we have $\mathsf{nf}(\sigma_2) \notin \vartheta$ or $Q(\mathsf{nf}(\sigma_2)) \in \mathcal{T}$. This assumption is also necessary to discard degenerate cases. For instance, let σ be $\forall\,(Q)\ \tau$. We have $(\alpha \geq \sigma)\ \sigma \sqsubseteq \alpha$ by I-Hyp. Without the assumption above, it could be given as input to the lemma. However, we cannot find a meaningful relation between the prefix $(\alpha \geq \sigma, Q)$ and the prefix $(\alpha \geq \sigma)$, as required by (4). In such a case, we cannot express (6) either.

Lemma 3.4.4 *We assume we have* $\mathsf{nf}(\sigma_2) \notin \vartheta$ *or* $Q(\mathsf{nf}(\sigma_2)) \in \mathcal{T}$. *We assume* $\mathsf{cf}(\sigma_1)$ *and* $\mathsf{cf}(\sigma_2)$ *have well-formed prefixes. If we have* $(Q)\ \sigma_1 \diamondsuit \sigma_2$ *and* $\mathsf{nf}(\sigma_1) \neq \bot$, $\mathsf{cf}(\sigma_1)$ *is* $\forall\,(Q_1)\ \tau_1$, *and* $Q_1 \# Q$, *then there exist an* α*-conversion* $\forall\,(Q_2)\ \tau_2$ *of* $\mathsf{cf}(\sigma_2)$ *and a substitution* θ *such that the following facts hold, writing* I *for* $\mathsf{dom}(Q_1/\tau_1)$.

$$(QQ_2)\ \theta(\tau_1) \equiv \tau_2 \qquad \mathsf{dom}(\theta) \subseteq I \qquad \theta(I) \subseteq \mathsf{dom}(Q) \cup \mathsf{dom}(Q_2/\tau_2)$$

$$QQ_1 \diamondsuit_\ell^{\mathsf{dom}(Q) \cup I} QQ_2\underline{\theta}$$

The proof is by induction on the derivation of $(Q)\ \sigma_1 \diamondsuit \sigma_2$.
See details in the Appendix (page 270).

As explained above, Lemma 3.4.4 is a main result for proving the equivalence between the relations $\sqsubseteq^I$ and $\sqsubseteq^I_\ell$. Actually, a first corollary of Lemma 3.4.4 is Property 3.4.5.i below. Then the expected equivalence (Property 3.4.5.ii) is a direct consequence.

Properties 3.4.5 *We have the following properties:*
i) *If* $(\emptyset)\ \forall\,(Q_1)\ \nabla_I \diamondsuit \forall\,(Q_2)\ \nabla_I$, *then* $Q_1 \diamondsuit^I_\ell Q_2$.
ii) *We have* $Q_1 \diamondsuit^I Q_2$ *iff* $Q_1 \diamondsuit^I_\ell Q_2$.

Proof: Property i: Assume that we have $\forall\,(Q_1)\ \nabla_I \diamondsuit \forall\,(Q_2)\ \nabla_I$. By Lemma 3.4.4, there exists a renaming ϕ and a substitution θ such that the following hold:

$$(\phi(Q_2))\ \theta(\nabla_I) \equiv \phi(\nabla_I)\ (\underline{\mathbf{1}}) \qquad J \triangleq \mathsf{dom}(Q_1/I)\ (\underline{\mathbf{2}}) \qquad \mathsf{dom}(\theta) \subseteq J$$

$$\theta(J) \subseteq \mathsf{dom}(\phi(Q_2)/\phi(I)) \qquad Q_1 \diamondsuit^J_\ell \phi(Q_2)\underline{\theta}\ (\underline{\mathbf{3}})$$

We have $I \subseteq J$ $(\underline{\mathbf{4}})$ from (2). Hence, we have $Q_1 \diamondsuit^I_\ell \phi(Q_2)\underline{\theta}$ $(\underline{\mathbf{5}})$ by Property 3.4.2.i, (4) and (3). By Property 1.5.11.vii (page 54) and (1), we have $\widehat{\phi(Q_2)} \circ \theta(\nabla_I) = \widehat{\phi(Q_2)} \circ \phi(\nabla_I)$. This implies that $\widehat{\phi(Q_2)} \circ \theta$ and $\widehat{\phi(Q_2)} \circ \phi$ are the same substitution on I. Hence, $\phi(Q_2)\underline{(\widehat{\phi(Q_2)} \circ \theta)} \equiv^I_\ell \phi(Q_2)\underline{(\widehat{\phi(Q_2)} \circ \phi)}$ $(\underline{\mathbf{6}})$ holds by PE-Free. Moreover, $\phi(Q_2)\underline{\widehat{\phi(Q_2)} \circ \phi} \equiv^I_\ell \phi(Q_2)\underline{\phi}$ $(\underline{\mathbf{7}})$ holds by PE-Mono*. Besides, $Q_2 \equiv_\ell \phi(Q_2)\underline{\phi}$ $(\underline{\mathbf{8}})$ holds by Property 3.4.2.iv. Additionally, we have $\phi(Q_2)\underline{\theta} \equiv^I_\ell \phi(Q_2)\underline{\widehat{\phi(Q_2)} \circ \theta}$ $(\underline{\mathbf{9}})$ by PE-Mono*. By (5), (9), (6), (7), (8), and PE-Trans, we get $Q_1 \diamondsuit^I_\ell Q_2$. This is the expected result.
Property ii: Directly, we have $Q_1 \diamondsuit^I Q_2$. Hence, $\forall\,(Q_1)\ \nabla_I \diamondsuit \forall\,(Q_2)\ \nabla_I$ holds by definition. Hence by Property i, $Q_1 \diamondsuit^I_\ell Q_2$ holds. The converse is from Lemma 3.4.1. ■

We have shown that $\diamondsuit^I$ and $\diamondsuit^I_\ell$ are the same relation. In the rest of the document, we will use only the symbols $\equiv^I$, $\sqsubseteq^I$, and $\sqsubseteq^I$ to make the presentation lighter.

ML types as a particular case

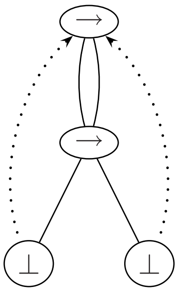
The next lemma shows that under an unconstrained prefix the instance relation corresponds to the instance relation of ML (in ML free variables are implicitly unconstrained in the prefix). As explained in Section 1.1, ML types can be injected in ML^{F} in the form $\forall\,(\alpha_1 \geq \bot) \ldots \forall\,(\alpha_n \geq \bot)\ \tau$. Hence, in the framework of ML^{F}, we call such types ML types:

Definition 3.4.6 Polytypes of the form $\forall\,(Q)\ \tau$ where Q is unconstrained are called ML types. □

Lemma 3.4.7 *We assume Q unconstrained and that σ_1 and σ_2 are ML types, closed under Q. Then $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ holds iff σ_2 is an instance of σ_1 in ML.*

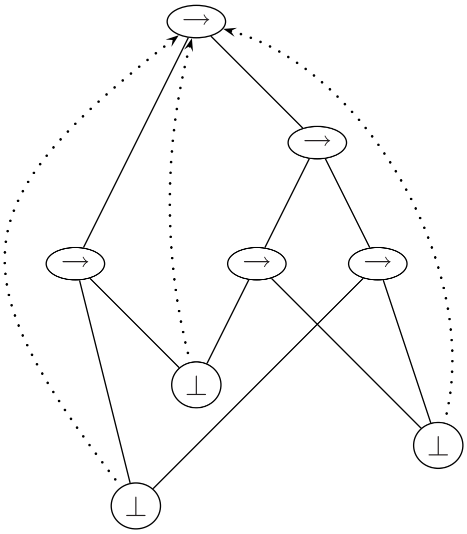
See proof in the Appendix (page 274).

Graphs representing ML types may only have binding arrows from nodes labelled $\bot$ to the top-level node. Besides, these arrows must be flexible. Here is an example of an ML type:



This graph represents the type $\forall\,(\alpha,\beta)\ (\alpha \to \beta) \to \alpha \to \beta$, that is, the type of $\lambda(f)\ \lambda(x)\ f\ x$, which we also call `app`. More precisely, we have represented $\forall\,(\alpha,\beta)$ $\forall\,(\gamma = \alpha \to \beta)\ \gamma \to \gamma$, which is equivalent. The middle node represents γ, the left-hand node is α and the right-hand node is β. The only bindings are from α and β to the top-level node.

Here is the type of a common function in ML:



We have represented the type $\forall(\alpha, \beta, \gamma)\ (\alpha \to \beta) \to (\beta \to \gamma) \to \alpha \to \gamma$, which is the type of the (reversed) composition function $\lambda(f)\ \lambda(g)\ \lambda(x)\ g\ (f\ x)$. (The type of the normal composition function gives a more intricate graph.)

3.5 Splitting prefixes

In this section we present the `split` algorithm, which takes a prefix Q and a set of variables $\bar{\alpha}$, and splits Q into two parts Q_1 and Q_2 such that the domain of Q_1 is the domain of Q useful for $\bar{\alpha}$ (see Definition 3.3.1). In order to show how the `split` algorithm is used, the following short story talks about generalization, which will be defined only in Chapter 5. This means that the following explanation is better understood by readers familiar with ML generalization, but should not frighten the others.

A short story of type generalization In ML, polymorphism is introduced by generalization. More precisely, if an expression a has type σ and if α is not free in the typing environment Γ, then the expression a can be given the type scheme $\forall(\alpha)\ \sigma$. This is possible for any type variable α not free in Γ, that is, if α and β are not free in Γ, a can be given the type $\forall(\alpha)\ \sigma$ as well as $\forall(\beta)\ \sigma$. As already mentioned in Section 1.2, the prefix in ML is implicit and unconstrained. Hence, generalizing a variable α consists mostly of taking α from the prefix and quantifying it in σ. The prefix before generalization is (Q, α), and the prefix after generalization is simply Q.

As for the type of a, it is σ before generalization, and $\forall\,(\alpha)\;\sigma$ after. Note that in ML, bindings in the prefix are mutually independent, which implies that the prefixes (α, β) and (β, α) are equivalent.

In ML^{F}, we follow the same idea for generalization: a binding is taken from the prefix and quantified in front of the type of the expression. Whereas bindings are unconstrained in ML, they are of the general form $(\alpha \diamond \sigma)$ in ML^{F}. Moreover, whereas bindings are mutually independent in ML, they can depend on each other in ML^{F}. For instance, in the prefix $Q = (\alpha \geq \bot, \beta = \forall\,(\gamma)\;\gamma \rightarrow \alpha)$, the binding for β depends on α because α is free in the bound of β. In ML, we could only have $(\alpha \geq \bot, \beta \geq \bot)$, which is equivalent to $(\beta \geq \bot, \alpha \geq \bot)$, as remarked above. In ML, if α is not free in Γ, we can generalize it. Similarly, if β is not free in Γ, we can generalize it. In ML^{F}, however, things are not so direct. Under prefix Q, we can generalize β (if β is not free in Γ), but we cannot generalize α directly because α is needed in the prefix, namely in the bound of β. As a consequence, the only way to generalize α is first to generalize β (if possible), then to generalize α. As a conclusion, generalization in ML^{F} must follow the dependencies of bindings. Therefore, when we wish to generalize as many variables as possible (according to a given typing environment Γ), we have to *split* the prefix Q in two parts. The first part contains all bindings of variables free in Γ, as well as bindings they depend on. The second part contains all remaining bindings, that is, type variables that do not appear in Γ, directly or indirectly. In this section, we describe this operation, the `split` algorithm. Note that, as mentioned above, splitting is trivial in ML since bindings do not depend on each other.

In the following definition, we write $q \xrightarrow{1} (Q_1, Q_2)$ for the pair $(Q_1 q, Q_2)$ and $q \xrightarrow{2} (Q_1, Q_2)$ for the pair $(Q_1, Q_2 q)$.

Definition 3.5.1 The `split` algorithm takes a closed prefix Q and a set $\bar{\alpha}$ included in $\mathsf{dom}(Q)$ and returns a pair of prefixes, which we write $Q{\uparrow}\bar{\alpha}$. It is defined inductively as follows:

$$\emptyset{\uparrow}\bar{\alpha} = (\emptyset, \emptyset) \qquad (Q, \alpha \diamond \sigma){\uparrow}\bar{\alpha} = \begin{cases} (\alpha \diamond \sigma) \xrightarrow{1} Q{\uparrow}(\bar{\alpha} - \alpha) \cup \mathsf{ftv}(\sigma) & \text{if } \alpha \in \bar{\alpha} \\ (\alpha \diamond \sigma) \xrightarrow{2} Q{\uparrow}\bar{\alpha} & \text{if } \alpha \notin \bar{\alpha} \end{cases} \qquad \square$$

It is obvious to check that this algorithm always terminates and never fails. The next lemma shows its correctness.

Lemma 3.5.2 *If $Q{\uparrow}\bar{\alpha}$ is the pair (Q_1, Q_2), then we have (i) $Q_1 Q_2 \approx Q$, (ii) $\bar{\alpha} \subseteq \mathsf{dom}(Q_1)$, and (iii) $\mathsf{dom}(Q_1/\bar{\alpha}) = \mathsf{dom}(Q_1)$.*

See proof in the Appendix (page 275).

3.6 Prefixes and Instantiation

This section gathers technical results that will be used mostly for unification. A main result is Lemma 3.6.4: it states that a judgment under a prefix Q still holds under all instances of Q. In ML, a substitution θ can be applied to a whole typing derivation. This can be viewed as instantiating all free type variables of the derivation. In particular, if σ_2 is an instance of σ_1 in ML, then $\theta(\sigma_2)$ is an instance of $\theta(\sigma_1)$. In ML^{F}, we state a more general result: typing derivations are stated under a given prefix, and instantiating the whole derivation amounts to instantiating the prefix. Thus, if σ_2 is an instance of σ_1 under prefix Q, then σ_2 must be an instance of σ_1 under any instance of Q (including monotypes instances, which represent substitutions). Hence, Lemma 3.6.4 entails that typing judgments are stable under substitution.

The next property extends the interface of an instantiation. For example, it takes a derivation of $Q \sqsubseteq^I Q'$ as an input, and provides a derivation of $Q \sqsubseteq^J Q'\underline{\theta}$ as an output, with $I \subseteq J$. This result is used for showing the completeness of the unification algorithm (to be found in Chapter 4).

Lemma 3.6.1 *Assume we have* $Q \Diamond^I Q'$*. Let* J *be* $\mathsf{dom}(Q/I)$ *and* ϕ *be a renaming of* $\mathsf{dom}(Q')$*, disjoint from* I*. Then there exists a substitution* θ *such that* $Q \Diamond^J \phi(Q')\underline{\theta}$ *and* $\mathsf{dom}(\theta) \subseteq J - I$ *hold.*

Proof: By Definition 3.2.1, we have $\forall(Q)\ \nabla_I \Diamond \forall(Q')\ \nabla_I$. Alpha-converting the right-hand side with the renaming ϕ, and observing that ϕ is disjoint from I, we get $\forall(Q)\ \nabla_I \Diamond \forall(\phi(Q'))\ \nabla_I$. Let J be $\mathsf{dom}(Q/I)$. By Lemma 3.4.4, there exists an alpha-conversion $\forall(Q'')\ \tau$ of $\forall(\phi(Q'))\ \nabla_I$ $\mathbf{(\underline{1})}$ and a substitution θ' such that we have

$$(Q'')\ \theta'(\nabla_I) \equiv \tau\ (\underline{\mathbf{2}}) \qquad \mathsf{dom}(\theta') \subseteq J \qquad \theta'(J) \subseteq \mathsf{dom}(Q''/\tau) \qquad Q \Diamond^J Q''\underline{\theta'}\ (\underline{\mathbf{3}})$$

From (1), there exists a renaming ϕ' such that $Q'' = \phi'(\phi(Q'))$, that is, $Q'' = \phi' \circ \phi(Q')$, and $\tau = \phi'(\nabla_I)$ $\mathbf{(\underline{4})}$. Let θ'' be the substitution defined as ϕ' on I and as θ' elsewhere $\mathbf{(\underline{5})}$. From (2) and (4), we get $(Q'')\ \theta'(\nabla_I) \equiv \phi'(\nabla_I)$, hence by Property 1.5.11.vii (page 54), we have $\widehat{Q''} \circ \theta'(\nabla_I) = \widehat{Q''} \circ \phi'(\nabla_I)$, that is, $\widehat{Q''} \circ \theta'$ and $\widehat{Q''} \circ \phi'$ are equal on I $\mathbf{(\underline{6})}$. Therefore, we have $\widehat{Q''} \circ \theta' = \widehat{Q''} \circ \theta''$ from (5) and (6). Then from (3), we get $Q \Diamond^J Q''\underline{\theta''}$ by PE-Mono*. Hence, $\forall(Q)\ \nabla_J \Diamond \forall(Q'')\ \theta''(\nabla_J)$ $\mathbf{(\underline{7})}$ holds by Definition 3.2.1 and Eq-Mono*. Let θ be $\phi'^{\neg} \circ \theta''$ restricted to J. We note that $\forall(\phi(Q'))\ \theta(\nabla_J)$ is an alpha-conversion of $\forall(Q'')\ \theta''(\nabla_J)$. Hence, $\forall(Q)\ \nabla_J \Diamond \forall(\phi(Q')\underline{\theta})\ \nabla_J$ holds from (7). By Property 3.4.5.i, we get $Q \Diamond^J \phi(Q')\underline{\theta}$. Besides, $\mathsf{dom}(\theta) \subseteq J$ and θ is invariant on I, thus $\mathsf{dom}(\theta) \subseteq J - I$. This is the expected result. ■

The next property is a variant: we also extend the interface of an instantiation $Q \Diamond^I Q'$, but the extended interface is the entire domain $\mathsf{dom}(Q)$ instead of $\mathsf{dom}(Q/I)$.

Lemma 3.6.2 *If we have* $Q \Diamond^I Q'$*, then there exists a renaming* ϕ *on* $\mathsf{dom}(Q')$ *disjoint from* I*, a prefix* Q_0*, a substitution* θ *and a prefix* Q'_0 *such that we have*

$$Q \Diamond \phi(Q')Q_0 \qquad \mathsf{dom}(Q_0) \# I \qquad \phi(Q')Q_0 \equiv^I Q' \qquad Q_0 = \underline{\theta} Q'_0$$

$$\mathsf{dom}(Q'_0) \# \mathsf{dom}(Q/I)$$

See proof in the Appendix (page 275).

As claimed in the introduction of this chapter, one main result is Lemma 3.6.4, to be found next page. The following properties are used directly in the proof of the lemma.

Properties 3.6.3 *Let* $\Diamond$ *be* $\sqsubseteq$ *or* $\mathrel{\sqsubseteq\mkern-12mu-}$*. We have the following properties:*

- ***i)*** *If* $(Q, \alpha \diamond \sigma)\ \sigma_1 \Diamond \sigma_2$ *holds and* $\alpha \notin \mathsf{ftv}(\sigma_1) \cup \mathsf{ftv}(\sigma_2)$*, then* $(Q)\ \sigma_1 \Diamond \sigma_2$ *holds.*
- ***ii)*** *If* $(Q, \alpha \geq \tau, Q_0)\ \sigma_1 \Diamond \sigma_2$*, then* $(Q, \alpha = \tau, Q_0)\ \sigma_1 \Diamond \sigma_2$*, and conversely.*
- ***iii)*** *If* $(Q, \alpha \diamond \sigma, \alpha' = \alpha, Q_0)\ \sigma_1 \Diamond \sigma_2$*, then* $(Q, \alpha' \diamond \sigma, \alpha = \alpha', Q_0)\ \sigma_1 \Diamond \sigma_2$*.*
- ***iv)*** *Assume* $(Q)\ \sigma \equiv \sigma'$*. If* $(Q, \alpha \diamond \sigma, Q_0)\ \sigma_1 \Diamond \sigma_2$*, then* $(Q, \alpha \diamond \sigma', Q_0)\ \sigma_1 \Diamond \sigma_2$*.*
- ***v)*** *Assume* $(Q)\ \sigma \mathrel{\sqsubseteq\mkern-12mu-} \sigma'$*. If* $(Q, \alpha = \sigma, Q_0)\ \sigma_1 \Diamond \sigma_2$*, then* $(Q, \alpha = \sigma', Q_0)\ \sigma_1 \Diamond \sigma_2$*.*
- ***vi)*** *Assume* $(Q)\ \sigma \sqsubseteq \sigma'$*. If* $(Q, \alpha \geq \sigma, Q_0)\ \sigma_1 \Diamond \sigma_2$*, then* $(Q, \alpha \geq \sigma', Q_0)\ \sigma_1 \Diamond \sigma_2$*.*
- ***vii)*** *If* $(Q, \alpha \geq \sigma, Q_0)\ \sigma_1 \Diamond \sigma_2$*, then* $(Q, \alpha = \sigma, Q_0)\ \sigma_1 \Diamond \sigma_2$*.*

See proof in the Appendix (page 276).

Lemma 3.6.4 *Assume* σ_1 *and* σ_2 *are in* Σ_I*. If* $Q_1 \Diamond^I Q_2$ *and* $(Q_1)\ \sigma_1 \Diamond \sigma_2$ *hold, then we have* $(Q_2)\ \sigma_1 \Diamond \sigma_2$*.*

See proof in the Appendix (page 278).

The following properties are used to show the correctness of unification.

Properties 3.6.5 *If we have*

$$Q_1 \Diamond^I Q_2 \qquad \gamma \in I \qquad \mathsf{dom}(Q_1/\gamma) \subseteq I \qquad \sigma_1 \notin \mathcal{V}$$

then we have the following:

- ***i)*** *If* $(\gamma \geq \sigma_1) \in Q_1$ *and* $(\gamma \diamond \sigma_2) \in Q_2$*, then* $(Q_2)\ \sigma_1 \sqsubseteq \sigma_2$*.*
- ***ii)*** *If* $(\gamma = \sigma_1) \in Q_1$ *and* $(\gamma \diamond \sigma_2) \in Q_2$*, then* $(Q_2)\ \sigma_1 \mathrel{\sqsubseteq\mkern-12mu-} \sigma_2$*.*
- ***iii)*** *If* $(\gamma \geq \sigma_1) \in Q_1$*, then* $(Q_2)\ \sigma_1 \sqsubseteq Q_2(\gamma)$
- ***iv)*** *If* $(\gamma = \sigma_1) \in Q_1$*, then* $(Q_2)\ \sigma_1 \mathrel{\sqsubseteq\mkern-12mu-} Q_2(\gamma)$

Proof: Property i: By hypothesis, $(\gamma \geq \sigma_1) \in Q_1$. Hence, $(Q_1)\ \sigma_1 \sqsubseteq \gamma$ **(1)** holds by I-Hyp, and we have $\mathsf{ftv}(\sigma_1) \subseteq \mathsf{dom}(Q_1/\gamma)$. Since we have $\mathsf{dom}(Q_1/\gamma) \subseteq I$ by hypothesis, this gives $\mathsf{ftv}(\sigma_1) \subseteq I$. Lemma 3.6.4 and (1) gives $(Q_2)\ \sigma_1 \sqsubseteq \gamma$. By Corollary 2.3.4, there exists a derivation of $(Q_2)\ \sigma_1 \sqsubseteq \sigma_2$.

Property ii: It is similar, using A-Hyp instead of I-Hyp.

Properties iii and iv: Let $(\gamma \diamond \sigma_2)$ be the binding of γ in Q_2. By Property i or ii, we have $(Q_2)\ \sigma_1 \Diamond \sigma_2$ **(2)**. If $\sigma_2 \notin \vartheta$, then $Q_2(\gamma) = \sigma_2$ by definition, thus $(Q_2)\ \sigma_1 \Diamond Q_2(\gamma)$ holds. Otherwise, $\sigma_2 \in \vartheta$, thus $\sigma_2 \equiv \alpha$ and $Q_2[\gamma] = Q_2[\alpha]$ **(3)** by definition. Hence, we have $Q_2(\gamma) = Q_2(\alpha)$ by definition and (3). Besides, (2) becomes $(Q_2)\ \sigma_1 \Diamond \alpha$. By Property 3.2.3.i (page 103) or Property 3.2.3.ii (page 103), we get $(Q_2)\ \sigma_1 \Diamond Q_2(\alpha)$. This is the expected result. ■

The following lemma is used to prove the completeness of unification.

Lemma 3.6.6 *If we have* $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ *and* $(Q)\ \sigma_2 \sqsubseteq \sigma_3$ *and* $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma_3$*, then we have both* $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma_2$ *and* $(Q)\ \sigma_2 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma_3$*.*

See proof in the Appendix (page 278).

The following lemma is used in the proof of Theorem 5, which states that System F can be encoded into ML^{F}.

Lemma 3.6.7 *If* $\forall\,(\alpha)\ \sigma \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma'$ *holds, then* $\sigma' \equiv \forall\,(\alpha)\ \sigma''$ *and* $\sigma \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma''$*.*

See proof in the Appendix (page 279).

The following property is used only in the proof of Lemma 3.6.9.

Lemma 3.6.8 *If* $\forall\,(Q)\ \alpha/u = \bot$ *holds, then there exist* Q_1*,* Q_2*,* β*,* σ*,* u_1*, and* u_2 *such that* Q *is* $(Q_1, \beta \diamond \sigma, Q_2)$*,* $u = u_1u_2$*,* $\forall\,(Q_2)\ \alpha/u_1 = \beta$*, and* $\sigma/u_2 = \bot$*.*

See proof in the Appendix (page 280).

The following lemma is a key result for the soundness of the `abstraction-check` algorithm (Def. 4.2.1), which is used by the unification algorithm.

Lemma 3.6.9 *If we have the following:*

$$\sigma_2 \in \Sigma_I \qquad \sigma_2 \notin \mathcal{V} \qquad (Q)\ \sigma_1 \sqsubseteq \sigma_2 \qquad Q \sqsubseteq^I Q' \qquad (Q')\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma_2$$

then, we also have $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma_2$*.*

See proof in the Appendix (page 280).

The following property is only used in Lemma 4.6.4, which shows that unification in ML$^{\text{F}}$ implements usual first-order unification as a special case. Indeed, as said in the introduction this Chapter, page 101, prefixes can be viewed as extended substitutions. Then prefix instance corresponds to composing substitutions. More precisely, we say that θ_1 is more general than θ_2, which we can write $\theta_1 \sqsubseteq \theta_2$, whenever there exists θ such that $\theta_2 = \theta \circ \theta_1$. The following property extends such a result to prefixes.

Lemma 3.6.10 *If* $Q_1 \sqsubseteq^I Q_2$ *holds, then there exists a substitution* θ *such that* $\widehat{Q_2}$ *and* $\theta \circ \widehat{Q_1}$ *are equal on* I*.*

See proof in the Appendix (page 281).

The following lemma is used in showing subject reduction in ML$^{\text{F}}_{\star}$.

Lemma 3.6.11 *If we have* $(\emptyset)\ \forall\,(Q_1)\ \tau_{11} \to \tau_{12} \Diamond \forall\,(Q_2)\ \tau_{21} \to \tau_{22}$ *then,* $(\emptyset)\ \forall\,(Q_1)\ \tau_{11} \Diamond \forall\,(Q_2)\ \tau_{21}$ *and* $(\emptyset)\ \forall\,(Q_1)\ \tau_{12} \Diamond \forall\,(Q_2)\ \tau_{22}$ *hold.*

See proof in the Appendix (page 281).

Note that the above lemma is stated under an empty prefix $(\emptyset)$, and with any relation $\Diamond$ in $\{\equiv, \mathrel{-\!\!\!\sqsubset}, \sqsubseteq\}$. One could wonder whether the result still holds under a well-formed prefix Q. Actually, such a result is easy to prove for the relations $\equiv$ and $\sqsubseteq$. We do not know if it holds for $\mathrel{-\!\!\!\sqsubset}$. Fortunately, we only use it with $\mathrel{-\!\!\!\sqsubset}$ and under an empty prefix.

The following property is used only in the proof of the Recomposition Lemma (Lemma 3.6.13).

Lemma 3.6.12 *If we have* $Q \sqsubseteq Q_1Q_2$ *and* $\mathsf{dom}(Q) = \mathsf{dom}(Q/I)$ *and* $I \subseteq \mathsf{dom}(Q_1)$*, then* $\mathsf{dom}(Q_2/\,\mathsf{dom}(Q)) \subseteq \mathsf{dom}(\widehat{Q_2})$ *and* $\widehat{Q_2}(\mathsf{dom}(Q)) \subseteq \mathsf{dom}(Q_1)$*.*

See proof in the Appendix (page 281).

The recomposition lemma, to be found next, is a key result used to show the completeness of unification and, independently, of type inference. It states, basically, that generalization can commute with the instantiation of the prefix. In ML, we would have the following diagram, where $\Gamma \vdash a : \sigma$ means that the expression a has type σ under the typing environment Γ. We assume that α, β, and γ are not free in the typing environment Γ.

$$\begin{tikzcd}
\Gamma \vdash a : \alpha \to \alpha \arrow[r, "\sqsubseteq\ (\underline{\mathbf{1}})"] \arrow[d, "\textsc{Gen}(\alpha)"'] & \Gamma \vdash a : (\beta \to \gamma) \to (\beta \to \gamma) \arrow[d, dotted, "\textsc{Gen}(\beta, \gamma)"] \\
\Gamma \vdash a : \forall\,(\alpha)\ \alpha \to \alpha \arrow[r, dotted, "\sqsubseteq\ (\underline{\mathbf{2}})"'] & \Gamma \vdash a : \forall\,(\beta, \gamma)\ (\beta \to \gamma) \to (\beta \to \gamma)
\end{tikzcd}$$

The arrow $\textsc{Gen}(\alpha)$ means that we generalize type variable α. Similarly, $\textsc{Gen}(\beta, \gamma)$ means that we generalize β and γ. The instantiation (1) consists, in ML, in applying the substitution $[\beta \to \gamma/\alpha]$ to the whole judgment. In ML^{F}, it consists of instantiating the implicit prefix (α) into $(\beta, \gamma, \alpha = \beta \to \gamma)$ (by PE-Free, PI-Context-L, and I-Nil). The instantiation (2) simply instantiates $\forall\,(\alpha)\ \alpha \to \alpha$ into $\forall\,(\beta, \gamma)\ (\beta \to \gamma) \to (\beta \to \gamma)$; this is valid in ML and in ML^{F}. Using the same notations as in the recomposition lemma, we have the following diagram:

$$\begin{tikzcd}
(Q_1)\ \Gamma \vdash a : \tau \arrow[r, "\sqsubseteq"] \arrow[d, "\begin{array}{c} Q_1 \approx Q_a Q_b \\ \textsc{Gen}(Q_b) \end{array}"'] & (Q_2 Q_3)\ \Gamma \vdash a : \tau \arrow[d, dotted, "\textsc{Gen}(Q_3)"] \\
(Q_a)\ \Gamma \vdash a : \forall\,(Q_b)\ \tau \quad (\underline{\mathbf{3}}) \arrow[r, dotted, "\sqsubseteq"'] & (Q_2)\ \Gamma \vdash a : \forall\,(Q_3)\ \tau \quad (\underline{\mathbf{4}})
\end{tikzcd}$$

Actually, the recomposition lemma does not mention typing judgments, but only prefixes. This is due to the expressiveness of prefixes, which can capture most of the type information of typing judgments. Indeed, a typing judgment such as (3) can be equivalently written $(Q_a, \gamma \geq \forall\,(Q_b)\ \tau)\ \Gamma \vdash a : \gamma$ **(5)**. Similarly, (4) can be written $(Q_2, \gamma \geq \forall\,(Q_3)\ \tau)\ \Gamma \vdash a : \gamma$ **(6)**. We see that all the interesting information is put in the prefix. We now understand why the recomposition lemma does not mention typing judgments but uses prefixes such as in (5) and (6) instead.

Lemma 3.6.13 (Recomposition lemma) *If we have*

$$Q_1 \sqsubseteq^{I \cup J} Q_2 Q_3 \qquad (Q_a, Q_b) = Q_1 \!\uparrow\! I \qquad I \subseteq \mathsf{dom}(Q_2) \qquad \mathsf{ftv}(\tau) \subseteq I \cup J$$

then, for γ fresh (that is, γ not in $\mathsf{dom}(Q_1) \cup \mathsf{dom}(Q_2) \cup \mathsf{dom}(Q_3)$), we can derive $(Q_a, \gamma \geq \forall\,(Q_b)\ \tau) \sqsubseteq^{I \cup \{\gamma\}} (Q_2, \gamma \geq \forall\,(Q_3)\ \tau)$.

See proof in the Appendix (page 283).

Chapter 4

Unification

The power of ML lies on its first-order unification algorithm, which enables type inference with outer-polymorphism (or, polymorphism *a la ML*). In ML, every solvable unification problem has a principal solution, which is a substitution. Such a solution is always found by the algorithm if it exists: the ML unification algorithm is *complete*. Conversely, unification of second-order polymorphic types is undecidable in general, therefore any second-order unification algorithm cannot be sound and complete. In ML^{F}, types are second order, but unification remains first-order. Indeed, while second-order unification usually requires "guessing" polymorphic types, the ML^{F} abstraction relation is designed to prevent implicit introduction of polymorphism. Hence, only explicit type annotations can create new second-order polymorphic types, so that the ML^{F} unification algorithm does not have to guess second-order polymorphism. In this chapter, we define unification of types, and give a sound and complete unification algorithm.

The solution to a unification problem is not a substitution, as in ML, but a prefix. As explained earlier, prefixes can be viewed as extended substitutions. In ML, we usually consider that a unification problem consists in two types τ_1 and τ_2, and that the solution is a single substitution θ that unifies τ_1 and τ_2. In a slightly different presentation, we can consider that the unification problem consists in two types τ_1 and τ_2, *and a substitution* θ (intuitively, the "current" substitution). Then a solution is a substitution θ', instance of θ, which unifies τ_1 and τ_2. This means that there exists θ'' such that θ' is $\theta'' \circ \theta$. Note that θ'' is actually the unifier of $\theta(\tau_1)$ and $\theta(\tau_2)$. We have a similar presentation in ML^{F}: a unification problem consists in two types τ_1 and τ_2, and a prefix Q. Then a solution is a prefix Q', instance of Q, which unifies τ_1 and τ_2, that is, such that $(Q')\ \tau_1 \equiv \tau_2$ holds. This definition is given in Section 4.1, and we also prove that it is an extension of ML unification.

As remarked in Section 3.1, monotype prefixes contain more information than substitutions. In particular, they contain the history of unification. More precisely, a monotype prefix corresponds to a composition of elementary substitutions. Take for example the prefix, say Q, equal to $(\beta \geq \bot, \gamma = \beta, \alpha = \gamma)$, which is a unifier for $\alpha \to \gamma$ and $\gamma \to \beta$. Its extracted substitution $\widehat{Q}$, which unifies the given types, is $\alpha \mapsto \beta, \gamma \mapsto \beta$. Whereas the substitution, alone, does not give much information, the prefix Q shows the history: α has first been unified to γ, hence the binding $(\alpha = \gamma)$, then γ has been unified to β, hence the binding $(\gamma = \beta)$. In the usual implementation of ML unification, we find a similar case: α would be an alias for γ, and γ would be an alias for β. An important optimization consists of updating the links, in order to have α and γ aliases for β. Similarly, in ML$^{\text{F}}$, the prefix $(\beta \geq \bot, \gamma = \beta, \alpha = \beta)$ is equivalent to Q by PE-Context-L and Eq-Mono. We see that the optimization is not transparent in ML$^{\text{F}}$ and corresponds to an equivalence between prefixes. This example illustrates that monotype prefixes capture more information than substitutions, and are closer to the implementation. Actually, monotype prefixes are very similar to multi-equations [Rém92], since they provide the same information. However, ML$^{\text{F}}$ prefixes are richer than multi-equations because a type variable can be bound to a polytype. This is also why an auxiliary algorithm is needed, namely `update`, that updates the polytype bound of a variable with another bound. This algorithm may rearrange elements of the prefix to satisfy dependencies.

The auxiliary algorithms `abstraction-check`, `merge` and `update` are defined in Section 4.2. Then the unification algorithm, called `unify`, is given in Section 4.3. As unification remain first-order, the algorithm is very similar to the ML unification algorithm, with only a few extra cases to handle polytype bounds. Soundness, termination, and completeness of the unification algorithm are shown respectively in Sections 4.4, 4.5, and 4.6.

4.1 Definition

We first give a simple specification of the unifier of two types under a given initial prefix. As explained above, the unifier is not a substitution, but rather a prefix.

Definition 4.1.1 (Unification) A prefix Q' *unifies* τ_1 and τ_2 under Q if and only if $Q \sqsubseteq Q'$ and $(Q')\ \tau_1 \equiv \tau_2$ hold. ☐

Such a definition is an extension of ML unification, as shown by the following lemma.

Lemma 4.1.2 *Let θ be a substitution and Q an unconstrained prefix that binds variables of* $\mathsf{codom}(\theta)$*. If Q' unifies τ_1 and τ_2 under the prefix $Q\underline{\theta}$, then $\widehat{Q'}$ is a unifier of τ_1 and τ_2 in ML.*

Figure 4.1: `Abstraction-check` algorithm

We assume that $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ holds. We define $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-}^{?} \sigma_2$ as follows :

1. If $\mathsf{proj}(\forall\,(Q)\ \sigma_1)$ and $\mathsf{proj}(\forall\,(Q)\ \sigma_2)$ are not equal, then **return** false.
2. If $\widehat{Q}(\mathsf{nf}(\sigma_1))$ is a monotype, then **return** true.
3. If $\widehat{Q}(\mathsf{nf}(\sigma_2))$ is not in ϑ, then **return** true if and only if $X \notin w(\sigma_1) - w(\sigma_2)$.
4. If $\widehat{Q}(\mathsf{nf}(\sigma_2))$ is α, find $(\alpha = \sigma)$ in Q (if such a binding cannot be found, **return** false), and **return** $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-}^{?} \sigma$.

Proof: Direct consequence of Property 1.5.11.vii (page 54). ■

4.2 Auxiliary algorithms

The unification algorithm uses three auxiliary algorithms: the `abstraction-check` algorithm, the `update` algorithm, and the `merge` algorithm.

The `abstraction-check` algorithm checks that the operations performed on binders are all safe. For example, it prevents the instantiation of $\sigma_{\mathtt{id}}$ in the type $\forall\,(\alpha = \sigma_{\mathtt{id}})\ \alpha \to \alpha$, which may occur for example when unifying this type with $(\mathtt{int} \to \mathtt{int}) \to (\mathtt{int} \to \mathtt{int})$. Indeed, the former type represents a function that expects a polymorphic argument and must not be instantiated. Interestingly, the check can be done *a posteriori*, and need not be done each time a binder is moved.

Definition 4.2.1 The `abstraction-check` algorithm $(Q)\ \sigma \mathrel{\sqsubseteq\!\!\!\!\!-}^{?} \sigma'$ is defined in Figure 4.1. It takes a prefix Q and two polytypes σ and σ' such that $(Q)\ \sigma \sqsubseteq \sigma'$ holds and tells if $(Q)\ \sigma \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma'$ holds. □

Lemma 4.2.2 *Assume we have $(Q)\ \sigma_1 \sqsubseteq \sigma_2$. Then $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma_2$ iff $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-}^{?} \sigma_2$ returns true.*

Proof: The termination, completeness and soundness of the `abstraction-check` algorithm are shown independently.

Termination There is only one place where the algorithm calls itself recursively. All other steps immediately terminate. In that case, $\widehat{Q}(\mathsf{nf}(\sigma_2))$ is α and $(\alpha = \sigma) \in Q$. By definition of $\widehat{Q}$, we must have $\sigma \notin \mathcal{T}$. Hence, $\widehat{Q}(\mathsf{nf}(\sigma)) \notin \vartheta$, and in this case the call to $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-}^{?} \sigma$ terminates immediately.

Completeness Completeness is shown by induction on the recursive calls to the `abstraction-check` algorithm. We assume $(Q)\ \sigma_1 \mathrel{\dot\sqsubseteq} \sigma_2$ holds. We show that $(Q)\ \sigma_1 \mathrel{\dot\sqsubseteq}^{?} \sigma_2$ returns true. By Lemmas 2.1.3.i (page 65) and 2.7.8, we have $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$ $\mathbf{(\underline{1})}$ and $X \notin w(\sigma_1) - w(\sigma_2)$ $\mathbf{(\underline{2})}$ provided $\widehat{Q}(\mathsf{nf}(\sigma_2)) \notin \mathcal{V}$. From (1), we have $\mathsf{proj}(\forall\,(Q)\ \sigma_1) = \mathsf{proj}(\forall\,(Q)\ \sigma_2)$ $\mathbf{(\underline{3})}$.

- The first step does not return false thanks to (3).
- If $(Q)\ \sigma_1 \equiv \tau$ holds, then $\widehat{Q}(\mathsf{nf}(\sigma_1))$ is a monotype $\widehat{Q}(\tau)$ by Lemma 1.5.9. Hence, the second step returns true. We can now assume that $(Q)\ \sigma_1 \equiv \tau$ does not hold for any monotype τ. In particular, we can assume that $\sigma_1 \notin \mathcal{V}$ $\mathbf{(\underline{4})}$.
- If $\widehat{Q}(\mathsf{nf}(\sigma_2))$ is not in ϑ, then the algorithm returns true thanks to (2).
- Otherwise, $\widehat{Q}(\mathsf{nf}(\sigma_2))$ is α, thus $(Q)\ \sigma_1 \mathrel{\dot\sqsubseteq} \alpha$ $\mathbf{(\underline{5})}$ holds by Property 1.5.6.i (page 51) and Eq-Mono. Let $(\alpha \diamond \sigma)$ be the binding of α in Q. By (4), (5) and Corollary 2.3.4, we have a derivation of $(Q)\ \sigma_1 \mathrel{\dot\sqsubseteq} \sigma$ $\mathbf{(\underline{6})}$ and $\diamond$ is rigid. Hence, the algorithm finds $(\alpha = \sigma)$ in Q and calls $(Q)\ \sigma_1 \mathrel{\dot\sqsubseteq}^{?} \sigma$. By induction hypothesis and (6), the algorithm returns true. This is the expected result.

Soundness Soundness is shown by induction on the recursive calls to the algorithm. We assume $(Q)\ \sigma_1 \mathrel{\dot\sqsubseteq}^{?} \sigma_2$ returns true, and we must show that $(Q)\ \sigma_1 \mathrel{\dot\sqsubseteq} \sigma_2$ holds. By hypothesis, $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ $\mathbf{(\underline{7})}$ holds.

- Since the first step does not return false, we must have $\mathsf{proj}(\forall\,(Q)\ \sigma_1) = \mathsf{proj}(\forall\,(Q)\ \sigma_2)$, that is, $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$ $\mathbf{(\underline{8})}$.
- If the algorithm returns true at the second step, we must have $\widehat{Q}(\mathsf{nf}(\sigma_1)) = \tau$ for some type τ. Hence, we have $\mathsf{nf}(\sigma_1) = \tau'$, which means $\sigma_1 \in \mathcal{T}$. By Lemma 2.1.6 and (7), we get $(Q)\ \sigma_1 \equiv \sigma_2$, which implies $(Q)\ \sigma_1 \mathrel{\dot\sqsubseteq} \sigma_2$ by A-Equiv.
- If the algorithm returns true at the third step, we must have $\widehat{Q}(\mathsf{nf}(\sigma_2)) \notin \vartheta$ $\mathbf{(\underline{9})}$ and $X \notin w(\sigma_1) - w(\sigma_2)$ $\mathbf{(\underline{10})}$. By (7), (8), (10) and (9) and by Lemma 2.7.8, we have $(Q)\ \sigma_1 \mathrel{\dot\sqsubseteq} \sigma_2$. This is the expected result.
- Otherwise, the algorithm necessarily returns true at the recursive call of the fourth step and $\widehat{Q}(\mathsf{nf}(\sigma_2))$ is α $\mathbf{(\underline{11})}$. Hence, $(\alpha = \sigma) \in Q$ and $(Q)\ \sigma_1 \mathrel{\dot\sqsubseteq}^{?} \sigma$ returns true. By induction hypothesis, we have $(Q)\ \sigma_1 \mathrel{\dot\sqsubseteq} \sigma$. By A-Hyp, we have $(Q)\ \sigma \mathrel{\dot\sqsubseteq} \alpha$, thus $(Q)\ \sigma_1 \mathrel{\dot\sqsubseteq} \alpha$ holds by R-Trans. Finally, $(Q)\ \sigma \mathrel{\dot\sqsubseteq} \sigma_2$ holds by Lemmas 1.5.10, 1.5.6.i (page 51), and (11).

■

The `update` algorithm replaces the bound of a prefix binding by a new one. It reorders the bindings in the prefix if necessary to make the resulting prefix well-formed. The condition $\alpha \notin \mathsf{dom}(Q/\sigma)$ ensures that there is no circular dependency, just like the

Figure 4.2: `Update` algorithm

```
We assume $\alpha \in \mathsf{dom}(Q)$, $\mathsf{ftv}(\sigma) \subseteq \mathsf{dom}(Q)$ and $\alpha \notin \mathsf{dom}(Q/\sigma)$.
We define $Q \Leftarrow (\alpha \diamond \sigma)$ as follows :
    1. let $(Q_1, Q_2)$ be $Q\!\uparrow \mathsf{ftv}(\sigma)$ in
    2. let $(Q_2^a, \alpha \diamond' \sigma', Q_2^b)$ be $Q_2$ in
    3. If $\diamond'$ is $=$ and $(Q)\ \sigma' \boxminus^? \sigma$ is false, then fail.
    4. return $(Q_1 Q_2^a, \alpha \diamond \sigma, Q_2^b)$.
```

occur-check in ML. The algorithm fails if a rigid bound is updated by a strict instance, using the abstraction-check algorithm. This algorithm is defined in Figure 4.2.

For example, if Q is $(\alpha \geq \sigma_{\mathtt{id}}, \beta \geq \bot)$, then the update $Q \Leftarrow (\alpha = \beta \rightarrow \beta)$ returns the prefix $(\beta \geq \bot, \alpha = \beta \rightarrow \beta)$. Notice that the binders α and β have been reordered, so that the resulting prefix is well-formed. As another example, if Q is $(\alpha = \sigma_{\mathtt{id}}, \beta \geq \bot)$, then the update $Q \Leftarrow (\alpha = \beta \rightarrow \beta)$ fails because it tries to instantiate the rigid bound $\sigma_{\mathtt{id}}$ to a strict instance $\beta \rightarrow \beta$.

The following lemma will be used to show the equivalence between the unification algorithm and a variant that reorders the bindings of the input prefix and removes unused bindings.

Lemma 4.2.3 *If $Q_1 \Leftarrow (\alpha \diamond \sigma)$ is well defined, and if Q_2 is a rearrangement of Q_1, then $Q_2 \Leftarrow (\alpha \diamond \sigma)$ is a rearrangement of $Q_1 \Leftarrow (\alpha \diamond \sigma)$.*

The following lemma states the completeness of the `update` algorithm.

Lemma 4.2.4 *If we have*

$$\begin{array}{llll} Q_1 \sqsubseteq^I Q_2 & (\alpha \diamond \sigma) \in Q_1 & (Q_1)\ \sigma \sqsubseteq \sigma' & (Q_1)\ \sigma \boxminus \sigma' \textit{ if } \diamond \textit{ is rigid} \\ \alpha \notin \mathsf{dom}(Q_1/\sigma') & \mathsf{ftv}(\sigma') \cup \{\alpha\} \subseteq I & (Q_2)\ \sigma' \sqsubseteq \alpha & (Q_2)\ \sigma' \boxminus \alpha \textit{ if } \diamond' \textit{ is rigid} \end{array}$$

then, we have $(Q_1 \Leftarrow (\alpha \diamond' \sigma')) \sqsubseteq^I Q_2$, and the update is well-defined.

See proof in the Appendix (page 284).

This lemma states the soundness of the `update` algorithm.

Lemma 4.2.5 *If we have $(\alpha \diamond \sigma) \in Q$, $(Q)\ \sigma \sqsubseteq \sigma'$ and $(Q \Leftarrow (\alpha \diamond \sigma'))$ returns Q', then $Q \sqsubseteq Q'$.*

See proof in the Appendix (page 285).

The `merge` algorithm unifies two type variables, which must have the same bound. The resulting binding is flexible if and only if the two variables are flexible. Otherwise, the new binding is rigid. This algorithm cannot fail.

Definition 4.2.6 The `merge` algorithm $Q \Leftarrow \alpha \wedge \alpha'$ takes two variables α and α' and a prefix Q such that Q is $(Q_0, \alpha \diamond \sigma, Q_1, \alpha' \diamond' \sigma, Q_2)$ or Q is $(Q_0, \alpha' \diamond' \sigma, Q_1, \alpha \diamond \sigma, Q_2)$ and returns the prefix $(Q_0, \alpha \diamond'' \sigma, \alpha' = \alpha, Q_1 Q_2)$ where $\diamond''$ is flexible if both $\diamond$ and $\diamond'$ are flexible, and rigid otherwise. □

The two following lemmas state the soundness and completeness of the `merge` algorithm, respectively.

Lemma 4.2.7 *If* $Q \Leftarrow \alpha \wedge \alpha'$ *returns* Q'*, then* $Q \sqsubseteq Q'$ *and* $(Q')\ \alpha \equiv \alpha'$*.*

See proof in the Appendix (page 285).

Lemma 4.2.8 *If we have*

$$Q_1 \sqsubseteq^I Q_2 \qquad \alpha, \alpha' \in I \qquad (\alpha \diamond \sigma) \in Q_1 \qquad (\alpha' \diamond' \sigma) \in Q_1 \qquad (Q_2)\ \alpha \equiv \alpha'$$

then $(Q_1 \Leftarrow \alpha \wedge \alpha') \sqsubseteq^I Q_2$*.*

See proof in the Appendix (page 285).

4.3 Unification algorithm

The algorithm `unify` takes a prefix Q and two types τ and τ' and returns a prefix that unifies τ and τ' under Q, as described in Theorem 1 (page 137), or fails. In fact, the algorithm `unify` is recursively defined with an auxiliary unification algorithm `polyunify` for polytypes: `polyunify` takes a prefix Q and two type schemes σ_1 and σ_2 not in $\mathcal{V}$ and returns a pair (Q', σ') such that $Q \sqsubseteq Q'$ and $(Q')\ \sigma_1 \sqsubseteq \sigma'$ and $(Q')\ \sigma_2 \sqsubseteq \sigma'$ hold.

The algorithms `unify` and `polyunify` are described in Figures 4.3 and 4.4, respectively. For the sake of comparison with ML, think of the input prefix Q as a substitution given to `unify` and of the result prefix Q' as an instance of Q (*i.e.* a substitution of the form $Q'' \circ Q$) that unifies τ and τ'. Unification of polytypes essentially follows the general structure of first-order unification of monotypes. The main differences are that (i) the computation of the unifying substitution is replaced by the computation of a unifying prefix, (ii) additional work must be performed when a variable bound to a strict polytype (*i.e.* other than $\bot$ and not equivalent to a monotype) is being unified:

Figure 4.3: Unification algorithm (monotypes)

```
unify $(Q, \tau_1, \tau_2)$
— Proceeds by case analysis on $(\tau_1, \tau_2)$ :
Case $(\alpha, \alpha)$: return $Q$.
Case $(g\ \tau_1^1\ ..\ \tau_1^n, g\ \tau_2^1\ ..\ \tau_2^n)$:
    • let $Q^0$ be $Q$ in
    • let $Q^i$ be unify $(Q^{i-1}, \tau_1^{i-1}, \tau_2^{i-1})$ for $i \in 1..n$ in
    • return $Q^n$.
Case $(g_1\ \tau_1^1\ ..\ \tau_1^p, g_2\ \tau_2^1\ ..\ \tau_2^q)$ with $g_1 \neq g_2$: fail.
Case $(\alpha, \tau)$ or $(\tau, \alpha)$ with $(\alpha \diamond \sigma) \in Q$ and $\sigma \in \mathcal{V}$: return unify $(Q, \tau, \mathsf{nf}(\sigma))$.
Case $(\alpha, \tau)$ or $(\tau, \alpha)$ with $(\alpha \diamond \sigma) \in Q$, $\tau \notin \vartheta$, and $\sigma \notin \mathcal{V}$:
    • fail if $\alpha \in \mathsf{dom}(Q/\tau)$.
    • let $(Q', \_)$ be polyunify $(Q, \sigma, \tau)$ in
    • return $(Q') \Leftarrow (\alpha = \tau)$
Case $(\alpha_1, \alpha_2)$ with $(\alpha_1 \diamond_1 \sigma_1) \in Q$, $(\alpha_2 \diamond_2 \sigma_2) \in Q$, $\alpha_1 \neq \alpha_2$, and $\sigma_1$, $\sigma_2$ not in $\mathcal{V}$:
    • fail if $\alpha_1 \in \mathsf{dom}(Q/\sigma_2)$ or $\alpha_2 \in \mathsf{dom}(Q/\sigma_1)$.
    • let $(Q', \sigma_3)$ be polyunify $(Q, \sigma_1, \sigma_2)$ in
    • return $(Q') \Leftarrow (\alpha_1 \diamond_1 \sigma_3) \Leftarrow (\alpha_2 \diamond_2 \sigma_3) \Leftarrow \alpha_1 \wedge \alpha_2$.
```

its bound must be further unified (last case of `polyunify`) and the prefix must then be updated accordingly.

The following is a stability property: it shows that if a binding $(\alpha \diamond \sigma) \in Q$ is useless, regarding two types τ_1 and τ_2, (that is, $\alpha \notin \mathsf{dom}(Q/\tau_1, \tau_2)$), then the `unify` algorithm does not remove the binding, and does not use it either (it is still useless after unification). Like all results about the unification algorithm, it is two-folded: one statement concerns the unification of monotypes (`unify`); the other concerns the unification of polytypes (`polyunify`). We recall that the notation $\mathsf{dom}(Q/\tau_1, \tau_2)$ means $\mathsf{dom}(Q/\tau_1) \cup \mathsf{dom}(Q/\tau_2)$.

Properties 4.3.1 *Assume* $(\alpha \diamond \sigma) \in Q$.

- ***i)*** *If* `unify` (Q, τ_1, τ_2) *returns a prefix* Q'*, and if* $\alpha \notin \mathsf{dom}(Q/\tau_1, \tau_2)$*, then* $(\alpha \diamond \sigma) \in Q'$ *and* $\alpha \notin \mathsf{dom}(Q'/\tau_1, \tau_2)$.
- ***ii)*** *If* `polyunify` (Q, σ_1, σ_2) *returns* (Q', σ_3)*, and if* $\alpha \notin \mathsf{dom}(Q/\sigma_1, \sigma_2)$*, then* $(\alpha \diamond \sigma) \in Q'$ *and* $\alpha \notin \mathsf{dom}(Q'/\sigma_1, \sigma_2, \sigma_3)$.

See proof in the Appendix (page 286).

Figure 4.4: Unification algorithm (polytypes)

```
polyunify $(Q, \sigma_1, \sigma_2)$
— Rewrite $\sigma_1$ and $\sigma_2$ in constructed form, and then:
Case $(\bot, \sigma)$ or $(\sigma, \bot)$: return $(Q, \sigma)$
Case $(\forall\,(Q_1)\ \tau_1, \forall\,(Q_2)\ \tau_2)$ with $Q_1$, $Q_2$, and $Q$ having disjoint domains (which usually requires renaming $\sigma_1$ and $\sigma_2$)
    • let $Q_0$ be unify $(QQ_1Q_2, \tau_1, \tau_2)$ in
    • let $(Q_3, Q_4)$ be $Q_0 \uparrow \mathsf{dom}(Q)$ in
    • return $(Q_3, \forall\,(Q_4)\ \tau_1)$
```

The following property shows that the `unify` algorithm is stable under rearrangement of the prefix, and under addition of new (unused) bindings.

Properties 4.3.2 *Assume* $Q \approx Q_1Q_2$.

i) *If* $\mathsf{ftv}(\tau, \tau') \subseteq \mathsf{dom}(Q_1)$, *then* `unify` (Q_1, τ, τ') *returns* Q'_1 *if and only if* `unify` (Q, τ, τ') *returns a rearrangement of* Q'_1Q_2.

ii) *If* $\mathsf{ftv}(\sigma, \sigma') \subseteq \mathsf{dom}(Q_1)$, *then* `polyunify` (Q_1, σ, σ') *returns* Q'_1 *and* σ'' *if and only if* `polyunify` (Q, σ, σ') *returns a rearrangement of* Q'_1Q_2 *and a rearrangement of* σ''.

See proof in the Appendix (page 286).

4.4 Soundness of the algorithm

In this section, we show the soundness of the `unify` and `polyunify` algorithms. More precisely, we show that if `unify` returns a prefix, it is an instance of the initial prefix and it does unify the types given in input. As for `polyunify`, it also returns two elements : an instance of the initial prefix and a common instance of σ_1 and σ_2 under this prefix. This means that `polyunify` returns the least upper bound of σ_1 and σ_2.

Lemma 4.4.1 (Soundness of the unification algorithm) *These two results hold:*

(i) If `unify` (Q, τ_1, τ_2) *succeeds with* Q', *then* $Q \sqsubseteq Q'$ *and* $(Q')\ \tau_1 \equiv \tau_2$.

(ii) Assume $\sigma_1 \notin \mathcal{V}$ *and* $\sigma_2 \notin \mathcal{V}$. *If* `polyunify` (Q, σ_1, σ_2) *succeeds with* (Q', σ), *then* $Q \sqsubseteq Q'$, $(Q')\ \sigma_1 \sqsubseteq \sigma$ *and* $(Q')\ \sigma_2 \sqsubseteq \sigma$ *hold.*

Proof: By induction on the recursive calls to both algorithms. We show the result for `unify` by case analysis on (τ_1, τ_2):

∘ Case (α, α) is immediate.

∘ Case $(g\ \tau_1^1\ ..\ \tau_1^n, g\ \tau_2^1\ ..\ \tau_2^n)$: Let Q^0 be Q, and Q^i be $\texttt{unify}\ (Q^{i-1}, \tau_1^{i-1}, \tau_2^{i-1})$ for $i \in 1..n$. By definition, Q' is Q^n. By induction hypothesis, we have $Q^0 \sqsubseteq Q^1 \ldots \sqsubseteq Q^{n-1} \sqsubseteq Q'$. Besides, $(Q^i)\ \tau_1^{i-1} \equiv \tau_2^{i-1}$ holds for $i \in 1..n$. Hence, by Lemma 3.4.3, we get $(Q')\ \tau_1^{i-1} \equiv \tau_2^{i-1}$. By Property 1.5.11.viii (page 54), this implies $(Q')\ \tau_1 \equiv \tau_2$.

∘ Case $(g_1\ \ldots, g_2\ \ldots)$ is not possible since the algorithm succeeds by hypothesis.

∘ Case (α, τ) or (τ, α) with $(\alpha \diamond \sigma) \in Q$ and $\sigma \in \mathcal{V}$: By induction hypothesis, $\texttt{unify}$ $(Q, \tau, \mathsf{nf}(\sigma))$ returns Q' such that $(Q')\ \tau \equiv \mathsf{nf}(\sigma)$ **(1)** and $Q \sqsubseteq Q'$. Moreover, we have $(Q)\ \alpha \equiv \mathsf{nf}(\sigma)$ by Eq-Mono, thus $(Q')\ \alpha \equiv \mathsf{nf}(\sigma)$ **(2)** holds by Lemma 3.6.4. Then $(Q')\ \tau \equiv \alpha$ holds by R-Trans, (1), and (2). This is the expected result.

∘ Case (α, τ) or (τ, α): We have $(\alpha \diamond \sigma) \in Q$ **(3)**, $\tau \notin \mathcal{V}$, and $\sigma \notin \mathcal{V}$. Let (Q_0, σ') be $\texttt{polyunify}\ (Q, \sigma, \tau)$. By induction hypothesis, we have $Q \sqsubseteq Q_0$ **(4)**, $(Q_0)\ \sigma \sqsubseteq \sigma'$ **(5)** and $(Q_0)\ \tau \sqsubseteq \sigma'$. By Lemma 2.1.6, this gives $(Q_0)\ \tau \equiv \sigma'$ **(6)**. Hence, $(Q_0)\ \sigma \sqsubseteq \tau$ **(7)** holds by R-Trans, (5), and (6). We have $\alpha \notin \mathsf{dom}(Q/\sigma)$ by well-formedness of Q and (3). Besides, $\alpha \notin \mathsf{dom}(Q/\tau)$, otherwise the algorithm would fail on the first step. Hence, by Property 4.3.1.i, $(\alpha \diamond \sigma) \in Q_0$ **(8)** and $\alpha \notin \mathsf{dom}(Q_0/\tau)$ **(9)**. Let Q' be $Q_0 \Leftarrow (\alpha = \tau)$ **(10)**. This update is well-defined by (9) and succeeds by hypothesis (the algorithm succeeds). By Lemma 4.2.5, (7) and (8), we get $Q_0 \sqsubseteq Q'$ **(11)**. By PI-Trans, (4), and (11), we get $Q \sqsubseteq Q'$. As a consequence of (10), we have $(\alpha = \tau) \in Q'$. Hence, $(Q')\ \alpha \equiv \tau$ holds by Eq-Mono.

∘ Case (α_1, α_2): We have $(\alpha_1 \diamond_1 \sigma_1) \in Q$ and $(\alpha_2 \diamond_2 \sigma_2) \in Q$. Moreover, σ_1 and σ_2 are not in $\mathcal{V}$ and $\alpha_1 \neq \alpha_2$. Let (Q_0, σ_3) be $\texttt{polyunify}\ (Q, \sigma_1, \sigma_2)$. By induction hypothesis, we have $Q \sqsubseteq Q_0$ **(12)** and $(Q_0)\ \sigma_1 \sqsubseteq \sigma_3$ **(13)** as well as $(Q_0)\ \sigma_2 \sqsubseteq \sigma_3$ **(14)**. We have $\alpha_1 \notin \mathsf{dom}(Q/\sigma_1)$ and $\alpha_1 \notin \mathsf{dom}(Q/\sigma_2)$, otherwise the algorithm would fail on the first step. Hence, by Property 4.3.1.i, $(\alpha_1 \diamond_1 \sigma_1) \in Q_0$ **(15)**, $\alpha_1 \notin \mathsf{dom}(Q_0/\sigma_3)$ **(16)** and $\alpha_2 \notin \mathsf{dom}(Q_0/\sigma_3)$ **(17)**. Let Q_1 be $Q_0 \Leftarrow (\alpha_1 \diamond_1 \sigma_3)$, Q_2 be $Q_1 \Leftarrow (\alpha_2 \diamond_2 \sigma_3)$, and Q' be $Q_2 \Leftarrow \alpha_1 \wedge \alpha_2$. The updates are well-defined by (16) and (17). By Lemma 4.2.5, (13) and (15), we get $Q_0 \sqsubseteq Q_1$ **(18)**. By Lemma 3.6.4, (18), and (14), we get $(Q_1)\ \sigma_2 \sqsubseteq \sigma_3$ **(19)**. We show that $(\alpha_2 \diamond_2 \sigma_2) \in Q_0$ as we showed (15). thus $(\alpha_2 \diamond_2 \sigma_2) \in Q_1$ **(20)** (the update does not modify the binding of α_2). Hence, by Lemma 4.2.5, (19) and (20), we get $Q_1 \sqsubseteq Q_2$ **(21)**. Finally, we have $Q_2 \sqsubseteq Q'$ **(22)** and $(Q')\ \alpha_1 \equiv \alpha_2$ by Lemma 4.2.7. By PI-Trans, (12), (18), (21), and (22), we get $Q \sqsubseteq Q'$.

We show the result for $\texttt{polyunify}$ by case analysis on the constructed forms of (σ_1, σ_2):

∘ Case $(\bot, \sigma)$ or $(\sigma, \bot)$: We return (Q, σ), thus the expected result holds by Property 1.5.13.i (page 55), and by rules PE-Refl, Eq-Refl and I-Bot.

∘ Case $\forall (Q_1)\ \tau_1$, $\forall (Q_2)\ \tau_2$: Let $Q_0 = \texttt{unify}\ (QQ_1Q_2, \tau_1, \tau_2)$. By induction hypothesis, Q_0 is such that $QQ_1Q_2 \sqsubseteq Q_0$ **(1)** and $(Q_0)\ \tau_1 \equiv \tau_2$ **(2)**. Let (Q_3, Q_4) be $Q_0 {\uparrow} \mathsf{dom}(Q)$ **(3)**. The returned prefix is $Q' = Q_3$. By Lemma 3.5.2 and (3), Q_0 is a rearrangement of Q_3Q_4 **(4)**. Hence (1) gives $QQ_1Q_2 \sqsubseteq^{\mathsf{dom}(Q)} Q_3Q_4$ **(5)** by Property 3.2.2.ii (page 103) and Property 3.4.2.i (page 106). Since $\mathsf{dom}(Q_1Q_2) \# \mathsf{dom}(Q)$ by hypothesis and $\mathsf{dom}(Q_4) \# \mathsf{dom}(Q)$ **($\underline{6}$)** by Lemma 3.5.2 and (3), we get $Q \sqsubseteq Q_3$ **($\underline{7}$)** by PE-FREE and (5). Additionally, $\mathsf{ftv}(\sigma_1) \subseteq \mathsf{dom}(Q)$, thus $\mathsf{dom}(Q_4) \# \mathsf{ftv}(\sigma_1)$ **($\underline{8}$)** holds by (6). We have

$$\begin{array}{cccll}
 & \sigma_1 & \equiv & \forall\,(Q_1)\,\tau_1 & \text{by Property 1.5.13.i (page 55)} \\
(QQ_1Q_2) & \sigma_1 & \sqsubseteq & \tau_1 & \text{by I-D}\textsc{rop}^\star \\
(Q_3Q_4) & \sigma_1 & \sqsubseteq & \tau_1 & \text{by Lemma 3.6.4 and (5)} \\
(Q_3) & \forall\,(Q_4)\,\sigma_1 & \sqsubseteq & \forall\,(Q_4)\,\tau_1 & \text{by R-C}\textsc{ontext}\text{-R} \\
(Q_3) & \sigma_1 & \sqsubseteq & \forall\,(Q_4)\,\tau_1 & \text{by (8) and E}\textsc{q}\text{-F}\textsc{ree}\text{.}
\end{array}$$

Similarly, we can show that $(Q_3)\ \sigma_2 \sqsubseteq \forall\,(Q_4)\ \tau_2$ holds. Moreover, by (2) and (4), we have $(Q_3Q_4)\ \tau_1 \equiv \tau_2$, and we get $(Q_3)\ \forall\,(Q_4)\ \tau_1 \equiv \forall\,(Q_4)\ \tau_2$ by R-CONTEXT-R. To sum up, we have (7) as well as $(Q_3)\ \sigma_1 \sqsubseteq \forall\,(Q_4)\ \tau_1$ and $(Q_3)\ \sigma_2 \sqsubseteq \forall\,(Q_4)\ \tau_1$. This is the expected result. ■

4.5 Termination of the algorithm

In order to make the proof of termination easier, we consider a variant of the unification algorithm, composed of two sub-algorithms `unify'` and `polyunify'`. The body of `polyunify'` is the body of `polyunify`, where the occurrence of `unify` is replaced by `unify'`. The algorithm `unify'` is given in Figure 4.5.

The differences between `unify` and `unify'` are in the cases (α, τ) and (α_1, α_2): The recursive call to `polyunify` is given a prefix Q_1 in `unify'`, instead of Q in `unify`. The prefix Q_1 contains all the bindings of Q which do not depend on α (in the first case), or on α_1 or α_2 (in the second case). Intuitively all the bindings that depends on α or on α_1 or α_2 are useless during the unification of α and τ or of α_1 and α_2 (respectively). In `unify'`, we explicitly remove such bindings, thus the recursive call to `polyunify` is made with a "smaller" prefix. This helps to show termination. First, we show that `unify` and `unify'` are equivalent. Then we define sizes and weights associated to a unification problem. Finally, we show that the algorithm `unify'` calls itself recursively with a strictly smaller weight, thus it terminates.

Lemma 4.5.1 *The algorithms* `unify` *and* `unify'` *are equivalent. More precisely, the algorithm* `unify` (Q, τ_1, τ_2) *returns* Q' *if and only if the algorithm* `unify'` (Q, τ_1, τ_2) *returns a rearrangement of* Q'*.*

See proof in the Appendix (page 286).

Definition 4.5.2 We define $\lceil \sigma \rceil$ as the number of universal quantifiers appearing syntactically in σ (considering $\forall\,(\alpha_1 \diamond_1 \sigma_1, Q)\ \sigma$ as syntactic sugar for $\forall\,(\alpha_1 \diamond_1 \sigma_1)\ \forall\,(Q)\ \sigma$. Similarly, $\lceil(\alpha_1 \diamond_1 \sigma_1, \ldots \alpha_n \diamond_n \sigma_n)\rceil$ is $n + \lceil \sigma_1 \rceil + \ldots \lceil \sigma_n \rceil$. We also define $\#\tau$ as the size of the set $\mathsf{dom}(\tau)$. □

Figure 4.5: Unification algorithm (variant)

```
unify' $(Q, \tau_1, \tau_2)$
— Proceeds by case analysis on $(\tau_1, \tau_2)$ :
Case $(\alpha, \alpha)$ : return $Q$.
Case $(g\ \tau_1^1\ ..\ \tau_1^n, g\ \tau_2^1\ ..\ \tau_2^n)$ :
    • let $Q^0$ be $Q$ in
    • let $Q^i$ be unify' $(Q^{i-1}, \tau_1^{i-1}, \tau_2^{i-1})$ for $i \in 1..n$ in
    • return $Q^n$.
Case $(g_1\ \tau_1^1\ ..\ \tau_1^p, g_2\ \tau_2^1\ ..\ \tau_2^q)$ with $g_1 \neq g_2$ : fail
Case $(\alpha, \tau)$ or $(\tau, \alpha)$ with $(\alpha \diamond \sigma) \in Q$ and $\sigma \in \mathcal{V}$ :
    • return unify' $(Q, \tau, \mathsf{nf}(\sigma))$.
Case $(\alpha, \tau)$ or $(\tau, \alpha)$ with $(\alpha \diamond \sigma) \in Q$, $\tau \notin \vartheta$, and $\sigma \notin \mathcal{V}$ :
    • fail if $\alpha \in \mathsf{dom}(Q/\tau)$.
    • let $I$ be $\{\beta \in \mathsf{dom}(Q) \mid \beta \neq \alpha$ and $\alpha \notin \mathsf{dom}(Q/\beta)\}$
    • let $(Q_1, Q_2)$ be $Q{\uparrow}I$
    • let $(Q'_1, \_)$ be polyunify' $(Q_1, \sigma, \tau)$.
    • return $Q'_1 Q_2 \Leftarrow (\alpha = \tau)$.
Case $(\alpha_1, \alpha_2)$ with $(\alpha_1 \diamond_1 \sigma_1) \in Q$, $(\alpha_2 \diamond_2 \sigma_2) \in Q$, $\alpha_1 \neq \alpha_2$, and $\sigma_1, \sigma_2$ not in $\mathcal{V}$ :
    • fail if $\alpha_1 \in \mathsf{dom}(Q/\sigma_2)$ or $\alpha_2 \in \mathsf{dom}(Q/\sigma_1)$.
    • let $I$ be $\{\beta \in \mathsf{dom}(Q) \mid \beta \notin \{\alpha_1, \alpha_2\}$ and $\{\alpha_1, \alpha_2\} \# \mathsf{dom}(Q/\beta)\}$
    • let $(Q_1, Q_2)$ be $Q{\uparrow}I$
    • let $(Q'_1, \sigma_3)$ be polyunify' $(Q_1, \sigma_1, \sigma_2)$
    • return $Q'_1 Q_2 \Leftarrow (\alpha_1 \diamond_1 \sigma_3) \Leftarrow (\alpha_2 \diamond_2 \sigma_3) \Leftarrow \alpha_1 \wedge \alpha_2$
```

Note that we have $\#\tau > 0$ for any τ, while $\lceil \tau \rceil = 0$.

Properties 4.5.3

i) *For any* Q *and* τ*, we have* $\lceil \forall\,(Q)\ \tau \rceil = \lceil Q \rceil$.

ii) *For any prefix* Q_1Q_2*, we have* $\lceil Q_1Q_2 \rceil = \lceil Q_1 \rceil + \lceil Q_2 \rceil$.

iii) *If* $Q \approx Q'$*, then* $\lceil Q \rceil = \lceil Q' \rceil$.

iv) *If* `unify'` (Q, τ_1, τ_2) *returns* Q'*, then* $\lceil Q' \rceil \leqslant \lceil Q \rceil$.

v) *If* `polyunify'` (Q, σ_1, σ_2) *returns* (Q', σ_3)*, then* $\lceil Q' \rceil + \lceil \sigma_3 \rceil \leqslant \lceil Q \rceil + \lceil \sigma_1 \rceil + \lceil \sigma_2 \rceil$.

See proof in the Appendix (page 288).

Properties 4.5.3.iv and 4.5.3.v tell us an interesting story. Indeed, by definition, $\lceil Q \rceil$ and $\lceil \sigma \rceil$ are the number of quantifiers appearing in Q and σ. Hence, the two aforementioned properties show that the number of quantifiers only decreases during unification. This corroborates the intuition that we perform only first-order unification, and that we never guess polymorphism, that is, we never introduce new quantifiers.

Weights

We associate a weight to each call to `unify'` (Q, τ_1, τ_2) and `polyunify'` (Q, σ_1, σ_2). A weight is a triple in $I\!N^3$, with its natural lexicographic ordering.

Definition 4.5.4 The weight associated to `unify'` (Q, τ_1, τ_2) is

$$(2 \times \lceil Q \rceil\,,\ \ \#\tau_1 + \#\tau_2\,,\ \ card(\mathsf{dom}(Q/\tau_1)) + card(\mathsf{dom}(Q/\tau_2)))$$

The weight associated to `polyunify'` (Q, σ_1, σ_2) is

$$(1 + 2 \times (\lceil Q \rceil + \lceil \sigma_1 \rceil + \lceil \sigma_2 \rceil)\,,\ \ 0\,,\ \ 0)$$ □

Lemma 4.5.5 (Termination of unification) *The* `unify` *algorithm always terminates, either by failing or by returning a prefix.*

Proof: By Lemma 4.5.1, `unify` (Q, τ_1, τ_2) terminates if and only if `unify'` (Q, τ_1, τ_2) **(1)** terminates. Hence, we show termination for `unify'` and `polyunify'`. Actually, we show that in the body of (1), all recursive calls to `unify'` and `polyunify'` have strictly smaller weights than the weight of (1). Similarly, we show that in the body of `polyunify'` (Q, σ_1, σ_2) **(2)**, recursive calls to `unify'` and `polyunify'` have strictly smaller weights than the weight of (2). Termination follows. We proceed by case analysis.

◦ Case (α, α) has no recursive call.

◦ Case $(g\ \tau_1^1\ ..\ \tau_1^n, g\ \tau_2^1\ ..\ \tau_2^n)$: By Property 4.5.3.iv, $\lceil Q^i \rceil \leqslant \lceil Q \rceil$. Besides, $\#\tau_1^i < \#\tau_1$ and $\#\tau_2^i < \#\tau_2$. Hence, the weight of `unify` $(Q^i, \tau_1^i, \tau_2^i)$ is strictly smaller than the weight of (1).

◦ Case $(g_1\ \tau_1^1\ ..\ \tau_1^n, g_2\ \tau_2^1\ ..\ \tau_2^n)$: both calls fail.

◦ Case (α, τ) with $(\alpha = \sigma) \in Q$ and $\sigma \in \mathcal{V}$: We call `unify` $(Q, \tau, \mathsf{nf}(\sigma))$ **(3)**. By hypothesis, we have $\sigma \in \mathcal{V}$, which means that $\mathsf{nf}(\sigma)$ is a type variable β such that $\mathsf{dom}(Q/\alpha) = \mathsf{dom}(Q/\beta) \cup \{\alpha\}$. Hence, $card(\mathsf{dom}(Q/\alpha)) = card(\mathsf{dom}(Q/\beta)) + 1$ **(4)**. The weight of (1) is the triple

$$(2 \times \lceil Q \rceil, \#\alpha + \#\tau, card(\mathsf{dom}(Q/\alpha)) + card(\mathsf{dom}(Q/\tau)))$$

The weight of (3) is

$$(2 \times \lceil Q \rceil, \#\beta + \#\tau, card(\mathsf{dom}(Q/\beta)) + card(\mathsf{dom}(Q/\tau)))$$

By (4), and observing that $\#\beta = \#\alpha$, this weight is equal to

$$(2 \times \lceil Q \rceil, \#\alpha + \#\tau, card(\mathsf{dom}(Q/\alpha)) - 1 + card(\mathsf{dom}(Q/\tau)))$$

Hence, the weight of (3) is strictly smaller than the weight of (2).

∘ Case (α, τ): Let I be $\{\beta \in \mathsf{dom}(Q) \mid \beta \neq \alpha \text{ and } \alpha \notin \mathsf{dom}(Q/\beta)\}$. Let (Q_1, Q_2) be $Q{\uparrow}I$. There is a recursive call to `polyunify'` (Q_1, σ, τ) **(<u>5</u>)**. By Lemmas 3.5.2 and 4.5.3.ii, we have $\lceil Q \rceil = \lceil Q_1 \rceil + \lceil Q_2 \rceil$ **(<u>6</u>)** Besides, $\alpha \in \mathsf{dom}(Q_2)$, thus we have $\lceil Q_2 \rceil \geqslant 1 + \lceil \sigma \rceil$. This implies $\lceil Q_1 \rceil + 1 + \lceil \sigma \rceil \leqslant \lceil Q_1 \rceil + \lceil Q_2 \rceil$, that is, $\lceil Q_1 \rceil + 1 + \lceil \sigma \rceil \leqslant \lceil Q \rceil$ **(<u>7</u>)** by (6). The first element of the weight of (1) is $2 \times \lceil Q \rceil$. The first element of the weight of (5) is $w = 1 + 2 \times (\lceil Q_1 \rceil + \lceil \sigma \rceil + \lceil \tau \rceil)$, that is, $w = 1 + 2 \times (\lceil Q_1 \rceil + \lceil \sigma \rceil)$. Hence, we have $w < 2 \times (\lceil Q_1 \rceil + 1 + \lceil \sigma \rceil)$. By (7), we get $w < 2 \times \lceil Q \rceil$. This implies that the weight of (5) is strictly smaller than the weight of (1).

∘ Case (α_1, α_2): Let I be $\{\beta \in \mathsf{dom}(Q) \mid \beta \notin \{\alpha_1, \alpha_2\} \text{ and } \{\alpha_1, \alpha_2\} \# \mathsf{dom}(Q/\beta)\}$ and (Q_1, Q_2) be $Q{\uparrow}I$. There is a recursive call to `polyunify` $(Q_1, \sigma_1, \sigma_2)$ **(<u>8</u>)**. By Lemmas 3.5.2 and 4.5.3.ii, we have $\lceil Q \rceil = \lceil Q_1 \rceil + \lceil Q_2 \rceil$ **(<u>9</u>)**. Besides, $\alpha_1, \alpha_2 \in \mathsf{dom}(Q_2)$, thus $\lceil Q_2 \rceil \geqslant 2 + \lceil \sigma_1 \rceil + \lceil \sigma_2 \rceil$ **(<u>10</u>)**. Hence, $\lceil Q_1 \rceil + 2 + \lceil \sigma_1 \rceil + \lceil \sigma_2 \rceil \leqslant \lceil Q \rceil$ **(<u>11</u>)** holds from (9) and (10). The first element of the weight of (1) is $2 \times \lceil Q \rceil$. The first element of the weight of (8) is $w = 1 + 2 \times (\lceil Q_1 \rceil + \lceil \sigma_1 \rceil + \lceil \sigma_2 \rceil)$. We have $w < 2 \times (\lceil Q_1 \rceil + 2 + \lceil \sigma_1 \rceil + \lceil \sigma_2 \rceil)$. Hence, by (11), we get $w < 2 \times \lceil Q \rceil$. As a consequence, the weight of (8) is strictly smaller than the weight of (1).

In `polyunify'` (Q, σ_1, σ_2) **(<u>12</u>)**, we call `unify'` $(QQ_1Q_2, \tau_1, \tau_2)$ **(<u>13</u>)**. The first element of the weight of (12) is $1 + 2 \times (\lceil Q \rceil + \lceil \sigma_1 \rceil + \lceil \sigma_2 \rceil)$, that is, by Property 4.5.3.i, $1 + 2 \times (\lceil Q \rceil + \lceil Q_1 \rceil + \lceil Q_2 \rceil)$. By Property 4.5.3.ii, the first element of the weight of (13) is $2 \times (\lceil Q \rceil + \lceil Q_1 \rceil + \lceil Q_2 \rceil)$. Hence, the weight of (13) is strictly smaller than the weight of (12). ■

4.6 Completeness of the algorithm

In this section, we show that the unification algorithm is also complete: if the unification problem admits a solution, then the algorithm finds a solution, which is equivalent or better (that is, more general).

The following property is used in the proof of completeness.

Lemma 4.6.1 *If* σ_1 *and* σ_2 *are not in* $\mathcal{V}$*, and if* `polyunify` (Q, σ_1, σ_2) *returns* (Q', σ_3)*, then,* σ_3 *is not in* $\mathcal{V}$*.*

See proof in the Appendix (page 288).

The intuitive specification of completeness is the following: if there is a solution to a given unification problem (*e.g.* $(Q_2)\ \tau_1 \equiv \tau_2$ below), then unification succeeds *and* returns a better result (*e.g.* $Q_1' \sqsubseteq^I Q_2$ below).

Lemma 4.6.2 (Completeness of unification) *Let I be $\mathsf{dom}(Q_1)$. We assume that τ_1, τ_2, σ_1, and σ_2 are in Σ_I. Furthermore, we assume that $Q_1 \sqsubseteq^I Q_2$ holds.*

(i) If $(Q_2)\ \tau_1 \equiv \tau_2$ holds, then `unify` *(Q_1, τ_1, τ_2) succeeds with Q'_1 and we have $Q'_1 \sqsubseteq^I Q_2$.*

(ii) If σ_1, σ_2, and σ_3 are not in $\mathcal{V}$, and if we have $(Q_2)\ \sigma_1 \sqsubseteq \sigma_3$ and $(Q_2)\ \sigma_2 \sqsubseteq \sigma_3$, then `polyunify` *$(Q_1, \sigma_1, \sigma_2)$ succeeds with Q'_1 and σ'_3 such that $(Q'_1, \gamma \geq \sigma'_3) \sqsubseteq^{I \cup \{\gamma\}} (Q_2, \gamma \geq \sigma_3)$*

Proof: We prove both properties simultaneously by induction on the recursive calls to the algorithm.

The `unify` algorithm:

Let I be $\mathsf{dom}(Q_1)$ **(1)**. We have to show that the algorithm cannot fail, thanks to the hypotheses $(Q_2)\ \tau_1 \equiv \tau_2$ **(2)** and $Q_1 \sqsubseteq Q_2$ **(3)**, and that the returned prefix, Q'_1 is such that $Q'_1 \sqsubseteq^I Q_2$. We proceed by case analysis on (τ_1, τ_2).

∘ Case (α, α) is immediate.

∘ Case $(g\ \tau_1^1\ ..\ \tau_1^n, g\ \tau_2^1\ ..\ \tau_2^n)$: *We will shown this case by iteration (from 1 to n).* By Property 1.5.11.viii (page 54) applied to (2), we have $(Q_2)\ \tau_1^i \equiv \tau_2^i$ for all $i \in 1..n$. Let Q^0 be Q_1 and Q^i be `unify` $(Q^{i-1}, \tau_1^{i-1}, \tau_2^{i-1})$. We have $Q^0 \sqsubseteq Q_2$ by hypothesis. We prove by (local) induction on i, that there exists a renaming ϕ^i on $\mathsf{dom}(Q_2)$ disjoint from I, and a prefix Q_0^i such that $Q^i \sqsubseteq \phi^i(Q_2)Q_0^i$ holds and $\mathsf{dom}(Q_0^i)\ \#\ I$. This is obviously true for $i = 0$, taking $\phi^0 = \mathtt{id}$ and $Q_0^i = \emptyset$. By (local) induction hypothesis, we have $Q^{i-1} \sqsubseteq \phi^{i-1}(Q_2)Q_0^{i-1}$ **(4)**. By Property 3.4.2.iv (page 106), we have $Q_2 \equiv \phi^{i-1}(Q_2)\underline{\phi^{i-1}}$, thus $Q_2 \equiv^I \phi^{i-1}(Q_2)$ holds by Property 3.4.2.i (page 106) and PE-Free. Then $Q_2 \equiv^I \phi^{i-1}(Q_2)Q_0^{i-1}$ **(5)** holds by PE-Free. We have $(Q_2)\ \tau_1^i \equiv \tau_2^i$ **(6)**, thus $(\phi^{i-1}(Q_2)Q_0^{i-1})\ \tau_1^i \equiv \tau_2^i$ **(7)** holds by Lemma 3.6.4 applied to (5) and (6). Hence, by induction hypothesis, (4), and (7), Q^i is defined (that is, the algorithm succeeds) and we have $Q^i \sqsubseteq^I \phi^{i-1}(Q_2)Q_0^{i-1}$. By Lemma 3.6.2 (page 114), there exists a renaming ϕ disjoint from I and a prefix Q_0 such that we have $Q^i \sqsubseteq \phi(\phi^{i-1}(Q_2)Q_0^{i-1})Q_0$. Taking $\phi^i = \phi \circ \phi^{i-1}$ and $Q_0^i = \phi(Q_0^{i-1})Q_0$ gives the expected result.

By (local) induction, the result holds for n, hence, we have

$$Q'_1 = Q^n \sqsubseteq \phi^n(Q_2)Q_0^n \equiv^I Q_2$$

Finally, we have $Q'_1 \sqsubseteq^I Q_2$. This is the expected result.

∘ Case $(g_1\ \ldots, g_2\ \ldots)$ with $g_1 \neq g_2$: The algorithm fails. Indeed, according to Property 1.5.11.viii, it is not possible to find Q_2 such that $(Q_2)\ g_1\ \ldots \equiv g_2\ \ldots$, thus this case does not happen.

∘ Case (α, τ) or (τ, α) with $(\alpha \diamond \sigma) \in Q_1$ and $\sigma \in \mathcal{V}$: By hypothesis, $(Q_2)\ \tau \equiv \alpha$ **(8)** holds. We have $(Q_1)\ \alpha \equiv \mathsf{nf}(\sigma)$ **(9)** by Eq-Mono. Moreover, $\mathsf{ftv}(\mathsf{nf}(\sigma)) \subseteq \mathsf{dom}(Q_1)$ holds, thus we have $\mathsf{ftv}(\mathsf{nf}(\sigma)) \subseteq I$ from (1). Hence, by Lemma 3.6.4, (3), and (9), we get

$(Q_2)\ \alpha \equiv \mathsf{nf}(\sigma)$ **(10)**. Consequently, $(Q_2)\ \tau \equiv \mathsf{nf}(\sigma)$ holds by R-Trans, (10), and (8). Hence, we get the result by induction hypothesis applied to `unify` $(Q_1, \tau, \mathsf{nf}(\sigma))$.

○ Case (α, τ) or (τ, α): We have $(\alpha \diamond \sigma) \in Q_1$ **(11)**, as well as $\tau \notin \vartheta$ **(12)** and $\sigma \notin \mathcal{V}$ **(13)**. By hypothesis, $(Q_2)\ \tau \equiv \alpha$ **(14)** holds. In the first point, below, we show that the algorithm cannot fail in the first step of the case (α, τ). In the second point, we use the induction hypothesis and the completeness of the update algorithm (Lemma 4.2.4) to show the result. The case where the binding is rigid might fail because of the abstraction-check algorithm. It is not the case thanks to Lemma 3.6.9, as shown below.

- If $\alpha \in \mathsf{dom}(Q_1/\tau)$, then Q_1 is of the form $(Q_a, \alpha \diamond \sigma, Q_b)$ and $\alpha \in \mathsf{ftv}(\forall\,(Q_b)\ \tau)$ **(15)**. Since $\tau \notin \vartheta$, we have $\forall\,(Q_b)\ \tau \notin \mathcal{V}$ **(16)**. We can derive $(Q_1)\ \forall\,(Q_b)\ \tau \sqsubseteq \tau$ **(17)** by I-Drop*. Hence, $(Q_2)\ \forall\,(Q_b)\ \tau \sqsubseteq \tau$ holds by Lemma 3.6.4, (17), and (3). By (14), we get $(Q_2)\ \forall\,(Q_b)\ \tau \sqsubseteq \alpha$ **(18)**. By Property 2.1.7.ii (page 68), (15), and (18), we have $\forall\,(Q_b)\ \tau \equiv \alpha$, which is not possible by (16). Hence, $\alpha \notin \mathsf{dom}(Q_1/\tau)$ **(19)**, and the algorithm does not fail on the first step.
- From (14) and Corollary 1.5.10, we must have $\alpha \in \mathsf{dom}(\widehat{Q_2})$. Hence, there exists τ' and σ' such that $(\alpha = \sigma') \in Q_2$ **(20)** and $\mathsf{nf}(\sigma') = \tau'$. Then $(Q_2)\ \alpha \equiv \sigma'$ **(21)** holds by Eq-Mono and Property 1.5.6.i. Note that (11) gives $\alpha \in I$ **(22)** and $\mathsf{dom}(Q_1/\alpha) \subseteq I$ **(23)** by (1). By Property 3.6.5.i (page 114), (2), (22), (23), (13), (11), and (20), we have $(Q_2)\ \sigma \sqsubseteq \sigma'$ **(24)**, thus $(Q_2)\ \sigma \sqsubseteq \tau$ **(25)** holds by R-Trans, (24), (21), and (14). We have $(Q_2)\ \tau \sqsubseteq \tau$ **(26)** by Eq-Refl. By induction hypothesis, (3), (13), (12), (25), and (26), `polyunify` (Q_1, σ, τ) succeeds with (Q', σ_3) **(27)** such that $(Q', \gamma \geq \sigma_3) \sqsubseteq^{I \cup \{\gamma\}} (Q_2, \gamma \geq \tau)$. By Property 3.4.2.i (page 106) and PE-Free, we get $Q' \sqsubseteq^I Q_2$ **(28)**. By Property 4.3.1.i (page 125) and (19), we have $(\alpha \diamond \sigma) \in Q'$ **(29)** and $\alpha \notin \mathsf{dom}(Q'/\tau)$ **(30)**. By (14), A-Equiv, and I-Abstract, we get $(Q_2)\ \tau \mathrel{-\!\!\!\!\sqsubseteq} \alpha$ **(31)** and $(Q_2)\ \tau \sqsubseteq \alpha$ **(32)**. The Soundness Lemma (Lemma 4.4.1) applied to (27) give $(Q')\ \tau \sqsubseteq \sigma_3$ **(33)**, $(Q')\ \sigma \sqsubseteq \sigma_3$ **(34)**, and $Q_1 \sqsubseteq Q'$. By Lemma 2.1.6 and (33), we have $(Q')\ \sigma_3 \equiv \tau$ **(35)**, thus $(Q')\ \sigma \sqsubseteq \tau$ **(36)** holds by R-Trans, (35), and (34). If $\diamond$ is rigid, we can derive $(Q_2)\ \sigma \mathrel{-\!\!\!\!\sqsubseteq} \tau$ by Property 3.6.5.ii (page 114), thus $(Q')\ \sigma \mathrel{-\!\!\!\!\sqsubseteq} \tau$ **(37)** holds by Lemma 3.6.9, (28), (36), and (12). Let Q'_1 be $Q' \Leftarrow (\alpha = \tau)$. By Lemma 4.2.4 (page 123), (28), (29), (37), (36), (30), (32) and (31), Q'_1 is well defined and $Q'_1 \sqsubseteq^I Q_2$ holds. This is the expected result.

○ Case (α_1, α_2): We have $(\alpha_1 \diamond_1 \sigma_1) \in Q_1$ **(38)**, and $(\alpha_2 \diamond_2 \sigma_2) \in Q_1$ **(39)**. By hypothesis, $(Q_2)\ \alpha_1 \equiv \alpha_2$ **(40)** holds, $\sigma_1, \sigma_2 \notin \mathcal{V}$ **(41)**. By well-formedness of (3), Q_1 is closed. Hence, (38) and (39) imply $\mathsf{utv}(\sigma_1) \cup \mathsf{utv}(\sigma_2) \subseteq \mathsf{dom}(Q_1)$, that is, $\sigma_1, \sigma_2 \in \Sigma_I$ **(42)** by (1).

We show, in a first time, that the algorithm does not fail on its first step. Assume, by a way of contradiction, that $\alpha_2 \in \mathsf{dom}(Q_1/\sigma_1)$ holds. Then Q_1 is of the form $(Q_a, \alpha_2 \diamond_2 \sigma_2, Q_b)$ and $\alpha_2 \in \mathsf{ftv}(\forall\,(Q_b)\ \sigma_1)$ **(43)**. From (41), we have $\forall\,(Q_b)\ \sigma_1 \notin \mathcal{V}$ **(44)**. We can derive $(Q_1)\ \forall\,(Q_b)\ \sigma_1 \sqsubseteq \alpha_1$ by I-Drop*, and I-Hyp or A-Hyp. Hence, $(Q_2)\ \forall\,(Q_b)\ \sigma_1 \sqsubseteq \alpha_1$ holds by Lemma 3.6.4 and (3). With (40) and R-Trans, we get $(Q_2)\ \forall\,(Q_b)\ \sigma_1 \sqsubseteq$

α_2 **(45)**. By Property 2.1.7.ii (page 68), (43), and (45), we have $\forall\,(Q_b)\ \sigma_1 \equiv \alpha_2$, which is not possible by (44). Hence, the hypothesis $\alpha_2 \in \mathsf{dom}(Q_1/\sigma_1)$ is not true. Similarly, we can show $\alpha_1 \notin \mathsf{dom}(Q_1/\sigma_2)$ **(46)**. As a consequence, the algorithm does not fail on the first step.

We now gather the necessary intermediate results to use the completeness of the update algorithm (Lemma 4.2.4). This proof is quite similar to the case (α, τ) above. It is a little more intricate, though, because both bounds are type schemes (not monotypes), and so is their unifier. If $\diamond_1$ is rigid, we have $(Q_1)\ \sigma_1 \mathrel{-\mkern-14mu\sqsubseteq} \alpha_1$ **(47)** by A-Hyp, and $(Q_1)\ \sigma_1 \sqsubseteq \alpha_1$ **(48)** by I-Abstract. If $\diamond_1$ is flexible, we have (48) directly by I-Hyp. By Lemma 3.6.4, (47), (48), and (3), we get $(Q_2)\ \sigma_1 \mathrel{-\mkern-14mu\sqsubseteq} \alpha_1$ **(49)** (when $\diamond_1$ is rigid), and $(Q_2)\ \sigma_1 \sqsubseteq \alpha_1$ **(50)**. By Property 3.2.3.i, (49), and (41), we get $(Q_2)\ \sigma_1 \mathrel{-\mkern-14mu\sqsubseteq} Q_2(\alpha_1)$ **(51)** (when $\diamond_1$ is rigid). By Property 3.2.3.ii, (50), and (41), we get $(Q_2)\ \sigma_1 \sqsubseteq Q_2(\alpha_1)$ **(52)**. Similarly, we have $(Q_2)\ \sigma_2 \sqsubseteq Q_2(\alpha_2)$ **(53)**, and $(Q_2)\ \sigma_2 \mathrel{-\mkern-14mu\sqsubseteq} Q_2(\alpha_2)$ when $\diamond_2$ is rigid. By Property 2.2.2.v (page 69) and (40), we have $(Q_2)\ Q_2(\alpha_1) \equiv Q_2(\alpha_2)$ **(54)**. Let σ_3 be $Q_2(\alpha_1)$ **(55)**. Note that by Definition 2.2.1, we have $\sigma_3 \notin \mathcal{V}$ **(56)**. By (52) and (55), we have $(Q_2)\ \sigma_1 \sqsubseteq \sigma_3$ **(57)**. By (53), (55), and (54), we get $(Q_2)\ \sigma_2 \sqsubseteq \sigma_3$ **(58)**. By induction hypothesis, (3), (41), (56), (57), and (58), `polyunify` $(Q_1, \sigma_1, \sigma_2)$ succeeds with (Q', σ'_3) such that $(Q', \gamma{\geq}\sigma'_3) \sqsubseteq^{I \cup \{\gamma\}} (Q_2, \gamma{\geq}\sigma_3)$ holds. By Lemma 3.6.1 (page 113), there exists a renaming ϕ disjoint from $I \cup \{\gamma\}$ and a substitution θ such that $(Q', \gamma \geq \sigma'_3) \sqsubseteq^{J \cup \{\gamma\}} (\phi(Q_2), \gamma \geq \phi(\sigma_3), \underline{\theta})$ **(59)** holds, J is the domain $\mathsf{dom}(Q'/I \cup \mathsf{ftv}(\sigma'_3))$ **(60)**, and $\mathsf{dom}(\theta) \subseteq J - I$ **(61)**. From (59), PE-Comm, and (61), we get $(Q', \gamma \geq \sigma'_3) \sqsubseteq^{J \cup \{\gamma\}} (\phi(Q_2)\underline{\theta}, \gamma \geq \phi(\sigma_3))$ **(62)**. Let σ''_3 be $\phi(\sigma_3)$ **(63)** and Q'_2 be $\phi(Q_2)\underline{\theta}$ **(64)** Then (62) becomes $(Q', \gamma{\geq}\sigma'_3) \sqsubseteq^{J \cup \{\gamma\}} (Q'_2, \gamma{\geq}\sigma''_3)$ **(65)**. We get $Q' \sqsubseteq^{J} Q'_2$ **(66)** by Property 3.4.2.i (page 106) and PE-Free. By Lemma 4.6.1, we have $\sigma'_3 \notin \mathcal{V}$ **(67)**. By (60), we have $\mathsf{dom}(Q', \gamma \geq \sigma'_3/\gamma) \subseteq J \cup \{\gamma\}$ **(68)**. We have $(Q'_2)\ \sigma'_3 \sqsubseteq \sigma''_3$ **(69)** by Property 3.6.5.i (page 114), (65), (68), (67). Additionally, $Q_2 \equiv \phi(Q_2)\underline{\phi}$ **(70)** holds by Property 3.4.2.iv (page 106), which leads to $Q_2 \equiv^{I} \phi(Q_2)\underline{\theta}$ **(71)** by Property 3.4.2.i (page 106) and PE-Free. From (71) and (64), we get $Q_2 \equiv^{I} Q'_2$ **(72)**. By soundness (Lemma 4.4.1), we have $(Q')\ \sigma_1 \sqsubseteq \sigma'_3$ **(73)**. Hence, $(Q'_2)\ \sigma_1 \sqsubseteq \sigma'_3$ **(74)** holds by Lemma 3.6.4 and (66). By (46) and Property 4.3.1.i (page 125), we have $(\alpha_1 \diamond_1 \sigma_1) \in Q'$ **(75)** as well as $\alpha_1 \notin \mathsf{dom}(Q'/\sigma'_3)$ **(76)**. By Property 3.2.3.iii (page 103) and (55), we get $(Q'_2)\ \sigma''_3 \sqsubseteq \alpha_1$ **(77)**. By (77), (69), and R-Trans, we get $(Q'_2)\ \sigma'_3 \sqsubseteq \alpha_1$ **(78)**.

We temporarily consider, in this paragraph, that $\diamond_1$ is rigid. Then (51) holds, that is, $(Q_2)\ \sigma_1 \mathrel{-\mkern-14mu\sqsubseteq} \sigma_3$. Then by (70) and Lemma 3.6.4, we get $(Q'_2\underline{\phi})\ \sigma_1 \mathrel{-\mkern-14mu\sqsubseteq} \sigma_3$, thus we get $(Q'_2)\ \phi(\sigma_1) \mathrel{-\mkern-14mu\sqsubseteq} \phi(\sigma_3)$ by R-Context-R. Since ϕ is a renaming disjoint from I, and by (63) and (42), this gives $(Q'_2)\ \sigma_1 \mathrel{-\mkern-14mu\sqsubseteq} \sigma''_3$ **(79)** . By Lemma 3.6.6, (74), (69) and (79), we get $(Q'_2)\ \sigma_1 \mathrel{-\mkern-14mu\sqsubseteq} \sigma'_3$ **(80)** and $(Q'_2)\ \sigma'_3 \mathrel{-\mkern-14mu\sqsubseteq} \sigma''_3$ **(81)**. By Lemma 3.6.9, (60), (67), (73), (66), (80), we get $(Q')\ \sigma_1 \mathrel{-\mkern-14mu\sqsubseteq} \sigma'_3$ **(82)**. $(Q')\ \sigma_1 \mathrel{-\mkern-14mu\sqsubseteq} \alpha_1$ holds by A-Hyp and (75). Hence, $(Q'_2)\ \sigma_1 \mathrel{-\mkern-14mu\sqsubseteq} \alpha_1$ **(83)** holds by Lemma 3.6.4 and (66). By Property 2.2.2.i (page 69), we have $(Q'_2)\ \alpha_1 \equiv Q'_2[\alpha_1]$ **(84)**. Thus, we get $(Q'_2)\ \sigma_1 \mathrel{-\mkern-14mu\sqsubseteq} Q'_2[\alpha_1]$ **(85)** by R-Trans, (83), and (84). By Corollary 2.3.4, (85) and (41), the bound of $Q'_2[\alpha_1]$ is rigid. Hence, $(Q'_2)\ Q'_2(\alpha_1) \mathrel{-\mkern-14mu\sqsubseteq} Q'_2[\alpha_1]$ holds by A-Hyp. That is, $(Q'_2)\ \sigma''_3 \mathrel{-\mkern-14mu\sqsubseteq} \alpha_1$ **(86)** by (63), (84), and R-Trans. Then $(Q'_2)\ \sigma'_3 \mathrel{-\mkern-14mu\sqsubseteq} \alpha_1$ **(87)** holds by (81), (86), and R-Trans.

We get back to the general case. We have $\mathsf{ftv}(\sigma'_3) \subseteq J$ **(88)** from (60). By hypothesis, we have $\alpha_1 \in I$, thus $\alpha_1 \in J$ $(\underline{\mathbf{89}})$. Let Q_a be $Q' \Leftarrow (\alpha_1 \diamond_1 \sigma'_3)$, let Q_b be $Q_a \Leftarrow (\alpha_2 \diamond_2 \sigma'_3)$, and Q'_1 be $Q_b \Leftarrow \alpha_1 \wedge \alpha_2$. By Lemma 4.2.4 (page 123), (66), (75), (73), (82), (76), (88), (89), (78) and (87), Q_a is well-defined and we have $Q_a \sqsubseteq^J Q'_2$. Similarly, Q_b is well-defined and we have $Q_b \sqsubseteq^J Q'_2$. By Property 3.4.2.i (page 106), we get $Q_b \sqsubseteq^I Q'_2$, and by (72), we have $Q_b \sqsubseteq^I Q_2$. Then by Lemma 4.2.8 (page 124), we have $Q'_1 \sqsubseteq^I Q_2$. This is the expected result.

The `polyunify` algorithm:

By hypothesis, σ_1, σ_2, and σ_3 are not in $\mathcal{V}$, and we have $(Q_2)\ \sigma_1 \sqsubseteq \sigma_3$ $(\underline{\mathbf{1}})$ and $(Q_2)\ \sigma_2 \sqsubseteq \sigma_3$ $(\underline{\mathbf{2}})$. Besides, $Q_1 \sqsubseteq^I Q_2$ $(\underline{\mathbf{3}})$ holds with $I = \mathsf{dom}(Q_1)$. Hence, $I \subseteq \mathsf{dom}(Q_2)$ $(\underline{\mathbf{4}})$. We show that the algorithm does not fail and returns (Q'_1, σ'_3) such that $(Q'_1, \gamma \geq \sigma'_3) \sqsubseteq^{I \cup \{\gamma\}} (Q_2, \gamma \geq \sigma_3)$ holds. We proceed by case on (σ_1, σ_2).

∘ CASE $(\bot, \sigma)$ or $(\sigma, \bot)$: Then we return (Q_1, σ). We have $(Q_2)\ \sigma \sqsubseteq \sigma_3$ $(\underline{\mathbf{5}})$ by (1) or (2). By hypothesis, we have $\mathsf{utv}(\sigma) \subseteq \mathsf{dom}(Q_1)$, that is $\mathsf{utv}(\sigma) \subseteq I$. Thus, we have $(Q_1, \gamma \geq \sigma) \sqsubseteq^{I \cup \{\gamma\}} (Q_2, \gamma \geq \sigma)$ by Property 3.4.2.iii (page 106). Then $(Q_1, \gamma \geq \sigma) \sqsubseteq^{I \cup \{\gamma\}} (Q_2, \gamma \geq \sigma_3)$ holds by PI-CONTEXT-L, (5) and PI-TRANS.

∘ CASE $(\forall\,(P_1)\ \tau_1, \forall\,(P_2)\ \tau_2)$: By hypothesis, we have $\mathsf{dom}(Q_1), \mathsf{dom}(P_1)$ and $\mathsf{dom}(P_2)$ disjoint. We have to gather the intermediate results necessary to use the Recomposition Lemma (Lemma 3.6.13), which immediately gives the expected result, as shown below.

By Lemma 3.4.4, (1) and (2), there exist two substitutions θ_1 and θ_2 and two alpha-conversions of σ_3, which we write $\forall\,(P_3)\ \tau_3$ and $\forall\,(P_4)\ \tau_4$, such that we have

$$Q_2P_1 \sqsubseteq^{\mathsf{dom}(Q_2) \cup I_1} Q_2P_3\underline{\theta_1} \quad (\underline{\mathbf{6}}) \qquad\qquad Q_2P_2 \sqsubseteq^{\mathsf{dom}(Q_2) \cup I_2} Q_2P_4\underline{\theta_2}$$

$$I_1 \triangleq \mathsf{dom}(P_1 / \mathsf{ftv}(\tau_1)) \qquad\qquad I_2 \triangleq \mathsf{dom}(P_2 / \mathsf{ftv}(\tau_2))$$

$$(Q_2P_3)\ \theta_1(\tau_1) \equiv \tau_3\ (\underline{\mathbf{7}}) \qquad\qquad (Q_2P_4)\ \theta_2(\tau_2) \equiv \tau_4$$

By Lemma 3.6.2 (page 114) and (6), there exists a renaming ϕ_1 on $\mathsf{dom}(Q_2P_3\underline{\theta_1})$, disjoint from $\mathsf{dom}(Q_2) \cup I_1$, and a prefix Q_0 such that we have $Q_2P_1 \sqsubseteq \phi_1(Q_2P_3\underline{\theta_1})Q_0$. Note that $\phi_1(\tau_1)$ is τ_1. Since ϕ_1 is disjoint from $\mathsf{dom}(Q_2)$, we have $Q_2P_1 \sqsubseteq Q_2\phi_1(P_3\underline{\theta_1})Q_0$. We define P'_3 as $\phi_1(P_3)$, τ'_3 as $\phi_1(\tau_3)$, and θ'_1 as the substitution extracted from $\phi_1(\underline{\theta_1})$. The last judgment can now be written $Q_2P_1 \sqsubseteq Q_2P'_3\underline{\theta'_1}Q_0$ $(\underline{\mathbf{8}})$. By Property 1.7.2.i (page 59) and (7), we get $(Q_2P'_3)\ \theta'_1(\tau_1) \equiv \tau'_3$ $(\underline{\mathbf{9}})$. Similarly, there exists ϕ_2 disjoint from $\mathsf{dom}(Q_2) \cup I_2$ and Q'_0, and we define P'_4 as $\phi_2(P_4)$, τ'_4 as $\phi_2(\tau_4)$ and θ'_2 as the substitution extracted from $\phi_2(\underline{\theta_2})$, so that we have $Q_2P_2 \sqsubseteq Q_2P'_4\underline{\theta'_2}Q'_0$ $(\underline{\mathbf{10}})$ and $(Q_2P'_4)\ \theta'_2(\tau_2) \equiv \tau'_4$ $(\underline{\mathbf{11}})$. By definition, $\forall\,(P_4)\ \tau_4$ is an alpha-conversion of $\forall\,(P_3)\ \tau_3$, thus there exists a renaming ψ of $\mathsf{dom}(P_3)$ such that $P_4 = \psi(P_3)$ and $\tau_4 = \psi(\tau_3)$ hold. Let ψ' be a renaming mapping $\mathsf{dom}(P'_4)$ to fresh variables, (that is, outside $\mathsf{dom}(Q_1) \cup \mathsf{dom}(Q_2) \cup \mathsf{dom}(P'_3) \cup \mathsf{dom}(P_1) \cup \mathsf{dom}(P_2) \cup \mathsf{dom}(\underline{\theta'_2}Q'_0)$). Let ϕ be $\psi' \circ \phi_2 \circ \psi \circ \phi_1^{\neg}$, let P_5 be $\psi'(P'_4)$ and τ_5 be $\psi'(\tau'_4)$. We have $P_5 = \phi(P'_3)$ as well as $\tau_5 = \phi(\tau'_3)$ $(\underline{\mathbf{12}})$ Hence, $\forall\,(P_5)\ \tau_5$ is an alpha-renaming of $\forall\,(P'_3)\ \tau'_3$. By Property 3.4.2.iv (page 106), we have $Q_2P'_4 \equiv Q_2P_5\psi'$. By Property 3.4.2.iii (page 106), it leads to $Q_2P'_4\underline{\theta'_2}Q'_0 \equiv Q_2P_5\psi'\underline{\theta'_2}Q'_0$ $(\underline{\mathbf{13}})$. Let θ''_2 be $\psi' \circ \theta'_2$ $(\underline{\mathbf{14}})$. By PI-TRANS, (13), (10), and (14), we get $Q_2P_2 \sqsubseteq Q_2P_5\underline{\theta''_2}Q'_0$ $(\underline{\mathbf{15}})$. By

Property 1.7.2.i (page 59) and (11), we get $(Q_2P_5)\ \theta''_2(\tau_2) \equiv \tau_5$ **(<u>16</u>)**. By construction, we have $P'_3 \# P_5$.

By Property 3.4.2.iii (page 106) and (8), we get $Q_2P_1P_2 \sqsubseteq Q_2P'_3\underline{\theta'_1}Q_0P_2$, which gives by PE-Comm $Q_2P_1P_2 \sqsubseteq Q_2P_2P'_3\underline{\theta'_1}Q_0$ **(<u>17</u>)**. By Property 3.4.2.iii (page 106), (15), we get

$$Q_2P_2P'_3\underline{\theta'_1}Q_0 \sqsubseteq Q_2P_5\underline{\theta''_2}Q'_0P'_3\underline{\theta'_1}Q_0$$

which gives by PE-Comm

$$Q_2P_2P'_3\underline{\theta'_1}Q_0 \sqsubseteq Q_2P'_3P_5\underline{\theta'_1\theta''_2}Q_0Q'_0 \quad (\underline{\mathbf{18}})$$

By PI-Trans, (17) and (18), we have

$$Q_2P_1P_2 \sqsubseteq Q_2P'_3P_5\underline{\theta'_1\theta''_2}Q_0Q'_0 \quad (\underline{\mathbf{19}})$$

By Property 3.4.2.v (page 106), we get

$$Q_2P'_3P_5\underline{\theta'_1\theta''_2}Q_0Q'_0 \sqsubseteq Q_2P'_3\underline{\phi^\neg}\,\underline{\theta'_1\theta''_2}Q_0Q'_0 \quad (\underline{\mathbf{20}})$$

Hence, by PI-Trans, (19), and (20), we have

$$Q_2P_1P_2 \sqsubseteq Q_2P'_3\underline{\phi^\neg}\,\underline{\theta'_1\theta''_2}Q_0Q'_0 \quad (\underline{\mathbf{21}})$$

By Lemma 3.6.4, (16), and (20), we have $(Q_2P'_3\underline{\phi^\neg}\,\underline{\theta'_1\theta''_2}Q_0Q'_0)\ \theta''_2(\tau_2) \equiv \tau_5$ **(<u>22</u>)**. Moreover, we have

$$\begin{array}{llll}
(Q_2P'_3\underline{\phi^\neg}\,\underline{\theta'_1\theta''_2}Q_0Q'_0)\quad \tau_1 & \equiv & \theta'_1(\tau_1) & \text{by Eq-Mono} \\
 & \equiv^{(\underline{\mathbf{23}})} & \tau'_3 & \text{by (9)} \\
 & = & \phi^\neg(\tau_5) & \text{by (12)} \\
 & \equiv & \tau_5 & \text{by Eq-Mono} \\
 & \equiv & \theta''_2(\tau_2) & \text{by (22)} \\
 & \equiv & \tau_2 & \text{by Eq-Mono}
\end{array}$$

In summary, we have shown that we have $(Q_2P'_3\underline{\phi^\neg}\,\underline{\theta'_1\theta''_2}Q_0Q'_0)\ \tau_1 \equiv \tau_2$ (<u>**24**</u>). Additionally, $(Q_2)\ \forall\,(P'_3\underline{\phi^\neg}\,\underline{\theta'_1\theta''_2}Q_0Q'_0)\ \tau_1 \equiv \forall\,(P'_3\underline{\phi^\neg}\,\underline{\theta'_1\theta''_2}Q_0Q'_0)\ \tau'_3$ holds by (23) and R-Context-R, that is, $(Q_2)\ \forall\,(P'_3\underline{\phi^\neg}\,\underline{\theta'_1\theta''_2}Q_0Q'_0)\ \tau_1 \equiv \sigma_3$ **(<u>25</u>)**. Similarly, we have $(Q_2)\ \forall\,(P'_3\underline{\phi^\neg}\,\underline{\theta'_1\theta''_2}Q_0Q'_0)\ \tau_2 \equiv \sigma_3$. By Property 3.4.2.iii (page 106) and (3), we get $Q_1P_1P_2 \sqsubseteq Q_2P_1P_2$. Hence, we have $Q_1P_1P_2 \sqsubseteq Q_2P'_3\underline{\phi^\neg}\,\underline{\theta'_1\theta''_2}Q_0Q'_0$ **(<u>26</u>)** from (21). Hence, by (26), (24), and induction hypothesis, the call to `unify` $(Q_1P_1P_2, \tau_1, \tau_2)$ succeeds with Q_0 such that $Q_0 \sqsubseteq^{\mathsf{dom}(Q_1P_1P_2)} Q_2P'_3\underline{\phi^\neg}\,\underline{\theta'_1\theta''_2}Q_0Q'_0$ **(<u>27</u>)**.

Let (Q_3, Q_4) be $Q_0{\uparrow}\,\mathsf{dom}(Q_1)$, that is, $Q_0{\uparrow}I$. By Lemma 3.6.13, (27), and (4), $\mathsf{ftv}(\tau_1) \subseteq I \cup \mathsf{dom}(P_1)$, taking $J = \mathsf{dom}(P_1P_2)$, and by (25), we get $(Q_3, \gamma \geq \forall\,(Q_4)\ \tau_1) \sqsubseteq^{I\cup\{\gamma\}} (Q_2, \gamma \geq \sigma_3)$. The algorithm returns $(Q_3, \forall\,(Q_4)\ \tau_1)$, thus Q'_1 is Q_3 and σ'_3 is $\forall\,(Q_4)\ \tau_1$. Hence, we have shown that $(Q'_1, \gamma \geq \sigma'_3) \sqsubseteq^{I\cup\{\gamma\}} (Q_2, \gamma \geq \sigma_3)$ holds. This is the expected result. ■

Corollary 4.6.3 *Assume* $Q_1 \sqsubseteq^I Q_2$ *and* τ_1, τ_2 *are in* Σ_I*. If* $(Q_2)\ \tau_1 \equiv \tau_2$ *holds, then* `unify` (Q_1, τ_1, τ_2) *succeeds with* Q'_1 *such that* $Q'_1 \sqsubseteq^I Q_2$*.*

See proof in the Appendix (page 289).

Theorem 1 *For any prefix* Q *and monotypes* τ_1 *and* τ_2*,* `unify` (Q, τ_1, τ_2) *returns the smallest prefix (for the relation* $\sqsubseteq^{\mathsf{dom}(Q)}$*) that unifies* τ_1 *and* τ_2 *under* Q*, or fails if there exists no prefix* Q' *that unifies* τ_1 *and* τ_2 *under* Q*.*

This theorem is a direct consequence of the soundness of the unification algorithm (Lemma 4.4.1), of its completeness (Lemma 4.6.2), and of its proof of termination (Lemma 4.5.5). Indeed, given a unification problem, the unification algorithm either fails or returns a prefix Q (termination). It it fails, the unification problem has no solution (completeness). Otherwise, Q is a solution to the unification problem (soundness) and all other solutions are instances of Q (completeness).

First-order unification The next lemma shows that the unification algorithm for ML^{F} returns the same result as the unification algorithm for ML when the given unification problem is actually an ML unification problem.

Lemma 4.6.4 *Given two monotypes* τ_1 *and* τ_2*, and an unconstrained prefix* Q *such that* τ_1 *and* τ_2 *are closed under* Q*,* `unify` (Q, τ_1, τ_2) *returns* Q' *such that* $\widehat{Q'}$ *is the principal unifier of* τ_1 *and* τ_2*, or fails if* τ_1 *and* τ_2 *cannot be unified.*

Proof: Termination: By Lemma 4.5.5, we know that `unify` (Q, τ_1, τ_2) always terminates.

Soundness: If `unify` (Q, τ_1, τ_2) returns a prefix Q', then by Lemma 4.4.1, we have $Q \sqsubseteq Q'$ and $(Q')\ \tau_1 \equiv \tau_2$ **(1)**. Let θ be $\widehat{Q'}$. By Property 1.5.11.vii (page 54) and (1), we have $\theta(\tau_1) = \theta(\tau_2)$. Hence, θ is a unifier of τ_1 and τ_2.

Completeness: Assume there exists a substitution θ such that $\theta(\tau_1) = \theta(\tau_2)$. Then (θ) $\tau_1 \equiv \tau_2$ **(2)** holds by Eq-Mono. Let I be $\mathsf{ftv}(\sigma_1) \cup \mathsf{ftv}(\sigma_2)$. Let Q'' be an unconstrained prefix such that $\mathsf{ftv}(\theta(I)) \subseteq \mathsf{dom}(Q'')$. Then $Q \sqsubseteq^I Q''\underline{\theta}$ **(3)** holds by PE-Free, PI-Context-L, and I-Bot. By Corollary 4.6.3, (2), and (3), `unify` (Q, τ_1, τ_2) succeeds and returns Q' such that $Q' \sqsubseteq^I Q''\underline{\theta}$. By Lemma 3.6.10 (page 116), there exists θ' such that we have $\theta = \theta' \circ \widehat{Q'}$. This is the expected result. ■

Part II

The programming language

The expressiveness of types, which we studied in Part I, is richer than in ML: types contain first-class polymorphism. In this part, we explain how to use such an expressiveness of types in the framework of a programming language, ML^{F}, and we show that this language is safe, that is, well-typed programs do not go wrong. Type safety in ML^{F} is shown by the usual combination of *progress*, which states that a well-typed expression is either a value or can be further reduced, and *subject reduction*, which states that reduction preserves typings. Since ML^{F} aims at being an extension of ML, the syntax of ML^{F} contains the syntax of ML, along with the same reduction rules. Actually, ML^{F} expressions are exactly ML expressions. Moreover, it will be shown in Corollary 8.1.7 that ML^{F} without type annotations has the same set of typable terms as ML.

Fortunately, the power available in ML^{F} polymorphic types can be introduced thanks to *type annotations*, which are a set of primitives. Those primitives have no dynamic cost: they behave just like the identity, but their given types make it possible to introduce first-class polymorphism in programs. Of course, such primitives have types that, in general, could not be expressed in ML. Their types need the expressiveness available in ML^{F}. Since we are interested in showing subject reduction, we have to associate a reduction rule to each type annotation. However, the reduction rules associated to some type annotations would need a language of types even richer than ML^{F} types. In other words, it is not possible to associate a reduction rule to every type annotation using the ML^{F} types. Hence, we use a trick, called *oracles*. Oracles are a simple mark put on an expression, which intuitively stands for a suitable type annotation. Then reduction rules associated to type annotations are easily written using oracles. The language made of ML^{F} and oracles is called $\mathrm{ML}^{\mathrm{F}}_{\star}$. Although we wish to prove type-safety in ML^{F}, we have to show subject-reduction and progress in $\mathrm{ML}^{\mathrm{F}}_{\star}$ because type annotations create oracles by reduction. Note that, like type annotations, oracles have no dynamic cost: they just behave like the identity. Their role is only static. Indeed, they are used in replacement of type annotations to allow the introduction of first-class polymorphism. A key difference between oracles and type annotations, or equivalently between $\mathrm{ML}^{\mathrm{F}}_{\star}$ and ML^{F}, is that type annotations contain all the type information needed for type inference, whereas oracles do not contain type information, but only indicate places where an annotation should be inserted. As a consequence, type inference is possible in ML^{F}. On the contrary, it is probably undecidable in $\mathrm{ML}^{\mathrm{F}}_{\star}$, just as in System F, even though the undecidability of the latter does not imply the undecidability of the former. See Part III and in particular Chapter 10 for a more detailed discussion about this issue.

This part is organized as follows: we define ML^{F} and $\mathrm{ML}^{\mathrm{F}}_{\star}$ in Chapter 5, and we give their static and dynamic semantics. Type safety is shown for $\mathrm{ML}^{\mathrm{F}}_{\star}$ in Chapter 6, and type safety for ML^{F} is a direct consequence. A type inference algorithm for ML^{F} is shown sound and complete in Chapter 7. Finally, type annotations are introduced as a set of primitives in Chapter 8. Type annotations can be viewed as a bridge between ML^{F}, which has type inference, and $\mathrm{ML}^{\mathrm{F}}_{\star}$, which has oracles needed for subject reduction.

Chapter 5

Syntax and semantics

This chapter introduces the syntax of $\mathrm{ML}^{\mathrm{F}}_{\star}$, which is the syntax of ML with oracles (Section 5.1). Then its static semantics, that is, the typing rules of $\mathrm{ML}^{\mathrm{F}}_{\star}$ are given in Section 5.2. They are similar to ML typing rules, with the additional rule for oracles that allows the reversal of the abstraction relation. This is called *revelation*. Since implicit revelation would break type inference, oracles are not allowed in ML^{F}, for which we wish to build a type-inference algorithm. They will replaced by explicit type annotations, as described in Chapter 8. Like in ML, we provide a syntax-directed set of typing rules in Section 5.3. Whereas the syntax-directed presentation of ML tries to keep judgments as instantiated as possible, we try to keep $\mathrm{ML}^{\mathrm{F}}_{\star}$ judgments as generalized as possible. Indeed, polymorphism has to be kept deeply in type structures, in order to ensure principal types. Binders can always be extruded to outer levels afterwards. Finally, Section 5.4 gives the dynamic semantics of $\mathrm{ML}^{\mathrm{F}}_{\star}$, which is identical to that of ML. Oracles do not contribute to reduction. They are simply propagated to indicate the locations that need to be annotated in reduced expressions .

We also consider a variant of $\mathrm{ML}^{\mathrm{F}}_{\star}$, called $\mathrm{UML}^{\mathrm{F}}$ (unannotated ML^{F}), where oracles are all implicit. As expected ML^{F}, $\mathrm{ML}^{\mathrm{F}}_{\star}$, and $\mathrm{UML}^{\mathrm{F}}$ have the same set of typable terms, but full type inference is possible only in ML^{F}.

5.1 Syntax

We assume given a countable set of variables, written with letter x, and a countable set of constants $c \in \mathcal{C}$. Every constant c has an arity $|c|$. A constant is either a primitive f or a constructor C. The distinction between constructors and primitives lies in their dynamic semantics: primitives (such as $+$) are reduced when fully applied, while constructors (such as `cons`) represent data structures, and are only reduced by

Figure 5.1: Expressions of $\mathrm{ML}^{\mathrm{F}}_{\star}$

$a ::= x \mid c \mid \lambda(x)\ a \mid a\ a \mid \texttt{let}\ x = a\ \texttt{in}\ a$	Terms
$\quad\mid (a : \star)$	Oracles
$c ::= f \mid C$	Constants
$z ::= x \mid c$	Identifiers

other primitives (or pattern matching in a suitable extension of the language). We use letter z to refer to identifiers, *i.e.* either variables x or constants c.

Expressions of $\mathrm{ML}^{\mathrm{F}}_{\star}$, written with letter a, are described in Figure 5.1. Expressions are those of ML extended with *oracles*. An oracle, written $(a : \star)$ is a place holder for an implicit type annotation around the expression a. Equivalently, the oracles can be replaced by explicit type annotations before type inference. Explicit annotations $(a : \sigma)$, which are described in Chapter 8, are actually syntactic sugar for applications $(\sigma)\ a$ where (σ) are constants. Some examples given in the introduction use the notation $\lambda(x : \sigma)\ a$, which is not defined in Figure 5.1 because it is syntactic sugar for $\lambda(x)$ $\texttt{let}\ x = (x : \sigma)\ \texttt{in}\ a$. Similarly, $\lambda(x : \star)\ a$ means $\lambda(x)\ \texttt{let}\ x = (x : \star)\ \texttt{in}\ a$.

The language ML^{F} is the restriction of $\mathrm{ML}^{\mathrm{F}}_{\star}$ to expressions that do not contain oracles. Programmers are expected to write their programs in ML^{F}, so that type inference is possible. Reductions rules, however, may introduce oracles. More precisely, oracles are only introduced by type annotation primitives. Hence, the language $\mathrm{ML}^{\mathrm{F}}_{\star}$ is only needed to show subject reduction for programs that contain type annotations.

5.2 Static semantics

Typing environments (or typing contexts), written with letter Γ, are lists of assertions of the form $z : \sigma$ that bind each identifier at most once. We write $z : \sigma \in \Gamma$ to mean that z is bound in Γ. We note that the order of assertions is not significant in Γ since identifiers are bound at most once. We assume given an initial typing context Γ_0 mapping constants to closed polytypes. A typing environment Γ *extends* Γ_0 whenever for all $z : \sigma$ in Γ_0, we have $z : \sigma$ in Γ.

Typing judgments are of the form $(Q)\ \Gamma \vdash a : \sigma$. A tiny difference with ML is the presence of the prefix Q that assigns bounds to type variables appearing free in Γ or σ. By comparison, this prefix is left implicit in ML because all free type variables have the same (implicit) bound $\bot$. In ML^{F} and $\mathrm{ML}^{\mathrm{F}}_{\star}$, we require σ and all polytypes of Γ to be closed with respect to Q, that is, $\mathsf{utv}(\Gamma) \cup \mathsf{utv}(\sigma) \subseteq \mathsf{dom}(Q)$.

Figure 5.2: Typing rules for $\mathrm{ML^F}$ and $\mathrm{ML^F_\star}$

$$\textsc{Var}\quad \frac{z : \sigma \in \Gamma}{(Q)\ \Gamma \vdash z : \sigma} \qquad \textsc{App}\quad \frac{(Q)\ \Gamma \vdash a_1 : \tau_2 \to \tau_1 \qquad (Q)\ \Gamma \vdash a_2 : \tau_2}{(Q)\ \Gamma \vdash a_1\ a_2 : \tau_1} \qquad \textsc{Fun}\quad \frac{(Q)\ \Gamma, x : \tau_0 \vdash a : \tau}{(Q)\ \Gamma \vdash \lambda(x)\ a : \tau_0 \to \tau}$$

$$\textsc{Let}\quad \frac{\begin{array}{c}(Q)\ \Gamma \vdash a_1 : \sigma \\ (Q)\ \Gamma, x : \sigma \vdash a_2 : \tau\end{array}}{(Q)\ \Gamma \vdash \mathtt{let}\ x = a_1\ \mathtt{in}\ a_2 : \tau} \qquad \textsc{Gen}\quad \frac{(Q, \alpha \diamond \sigma)\ \Gamma \vdash a : \sigma' \qquad \alpha \notin \mathsf{ftv}(\Gamma)}{(Q)\ \Gamma \vdash a : \forall\,(\alpha \diamond \sigma)\ \sigma'}$$

$$\textsc{Inst}\quad \frac{(Q)\ \Gamma \vdash a : \sigma \qquad (Q)\ \sigma \sqsubseteq \sigma'}{(Q)\ \Gamma \vdash a : \sigma'} \qquad \textsc{Oracle}\quad \frac{(Q)\ \Gamma \vdash a : \sigma \qquad (Q)\ \sigma \mathrel{\exists} \sigma'}{(Q)\ \Gamma \vdash (a : \star) : \sigma'}$$

Typing rules The typing rules of $\mathrm{ML^F_\star}$ and $\mathrm{ML^F}$ are described in Figure 5.2. They correspond to the typing rules of ML modulo the richer types, the richer instance relation, and the explicit binding of free type variables in judgments (in the prefix). In addition, Rule Oracle allows for the *revelation* of polytypes, that is, the transformation of types along the inverse of the abstraction relation. (This rule would have no effect in ML where abstraction is the same as equivalence.) As explained in the introduction, we also consider $\mathrm{UML^F}$, a variant of $\mathrm{ML^F_\star}$ where oracles are left implicit. This amounts to replacing Rule Oracle by U-Oracle given below or, equivalently, combine Oracle with Inst into U-Inst.

$$\textsc{U-Oracle}\quad \frac{(Q)\ \Gamma \vdash a : \sigma \qquad (Q)\ \sigma \mathrel{\exists} \sigma'}{(Q)\ \Gamma \vdash a : \sigma'} \qquad \textsc{U-Inst}\quad \frac{(Q)\ \Gamma \vdash a : \sigma \qquad (Q)\ \sigma \mathrel{\exists\sqsubseteq\exists} \sigma'}{(Q)\ \Gamma \vdash a : \sigma'}$$

In Rule U-Inst, we have to allow revelation before *and* after instantiation because $\sqsubseteq$ and $\exists$ do not commute in general, under an arbitrary prefix. However, $\exists\sqsubseteq\exists\sqsubseteq$ happens to be equal[1] to $\exists\sqsubseteq\exists$. Thus, Rule U-Inst never needs to be used twice on the same expression.

As in ML, there is an important difference between rules Fun and Let: while typechecking their bodies, a let-bound variable can be assigned a polytype, but a λ-bound variable can only be assigned a monotype in Γ. Indeed, the latter must be guessed while the former can be inferred from the type of the bound expression. This

---

[1] This statement is not shown in this document. It is proved under an unconstrained prefix using the Diamond Lemma (Lemma 2.8.4).

restriction is essential to enable type inference. Notice that a λ-bound variable can refer to a polytype *abstractly* via a type variable α bound to a polytype σ in Q. However, this will not make it possible to take different instances of σ while typing the body of the abstraction, unless the polytype bound σ of α is first *revealed* by an oracle. Indeed, the only possible instances of α under a prefix Q that contains the binding $(\alpha = \sigma)$ are types equivalent to α under Q, as shown by Lemma 2.1.6. However, $(Q)\ \alpha \equiv \sigma$ does not hold (see Property 1.5.11.x (page 54)). Thus, if $x : \alpha$ is in the typing context Γ, the only way of typing x (modulo equivalence) is $(Q)\ \Gamma \vdash x : \alpha$, whereas $(Q)\ \Gamma \vdash x : \sigma$ is not derivable. Conversely, $(Q)\ \Gamma \vdash (x : \star) : \sigma$ is derivable, since $(Q)\ \alpha \sqsupseteq \sigma$ holds. This prevents, for example, $\lambda(x)\ x\ x$ from being typable in ML$^{\text{F}}$—see Example 6.2.12.

5.2.1 ML as a subset of ML$^{\text{F}}$

ML can be embedded into ML$^{\text{F}}$ by restricting all bounds in the prefix Q to be unconstrained. Rules Gen and Inst are then exactly those of ML, by Lemma 3.4.7. Hence, any closed program typable in ML is also typable in ML$^{\text{F}}$.

The converse will be shown in Section 8.1; that is, terms typable in ML$^{\text{F}}$ without primitives are also typable in ML.

5.2.2 Examples of typings

In this section, we consider three examples of typings:

- The first one illustrates that the canonical typing in ML$^{\text{F}}$ (or principal typing) of a term can be quite different from its typing in ML.
- The second example focuses on the role of rigid bindings in types: we introduce two constants whose type differ only by one rigid binding. In the type of the first constant, a single rigid binding is used. In the type of the second constant, the same rigid binding is duplicated. We compare the typing of expressions using either one of these constants.
- The third example considers the "app" property, which illustrates the compositionality of ML$^{\text{F}}$.

Example 5.2.9 This first example of typing illustrates the use of polytypes in typing derivations: we consider the simple expression K' defined by $\lambda(x)\ \lambda(y)\ y$. Following ML, one possible typing derivation is (recall that (α, β) stands for $(\alpha \geq \bot, \beta \geq \bot)$):

$$\textsc{Gen}\ \dfrac{\textsc{Fun}\ \dfrac{\textsc{Fun}\ \dfrac{(\alpha,\beta)\ x : \alpha, y : \beta \vdash y : \beta}{(\alpha,\beta)\ x : \alpha \vdash \lambda(y)\ y : \beta \to \beta}}{(\alpha,\beta)\ \vdash K' : \alpha \to (\beta \to \beta)}}{\vdash K' : \forall\,(\alpha,\beta)\ \alpha \to (\beta \to \beta)}$$

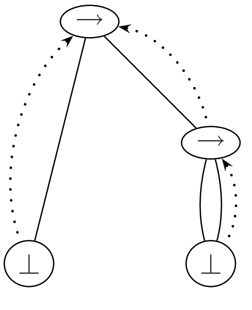
There is, however, another typing derivation that infers a more general type for K' in ML$^{\mathrm{F}}$ (for conciseness we write Q for $(\alpha, \beta \geq \sigma_{\mathtt{id}})$, and we recall that $\sigma_{\mathtt{id}}$ is $\forall\,(\alpha)\ \alpha \to \alpha$):

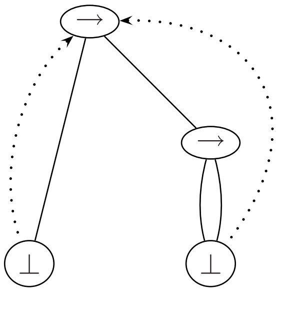
$$
\textsc{Gen}\ \dfrac{\textsc{Fun}\ \dfrac{\textsc{Inst}\ \dfrac{\textsc{Gen}\ \dfrac{\textsc{Fun}\ \dfrac{(Q,\gamma)\ x:\alpha, y:\gamma \vdash y:\gamma}{(Q,\gamma)\ x:\alpha \vdash \lambda(y)\ y:\gamma\to\gamma}}{(Q)\ x:\alpha \vdash \lambda(y)\ y:\sigma_{\mathtt{id}}} \qquad \dfrac{}{(Q)\ \sigma_{\mathtt{id}} \sqsubseteq \beta}\ \textsc{I-Hyp}}{(Q)\ x:\alpha \vdash \lambda(y)\ y:\beta}}{(Q)\ \vdash K':\alpha\to\beta}}{\vdash K' : \forall\,(Q)\ \alpha\to\beta}
$$

Notice that the polytype $\forall\,(Q)\ \alpha \to \beta$ is more general than $\forall\,(\alpha,\beta)\ \alpha \to (\beta \to \beta)$, which follows from Example 1.8.7. Actually, this derivation gives the principal type of K'. Satisfyingly, this is also the type inferred by the type inference algorithm (given in Chapter 7), and implemented in a prototype of ML$^{\mathrm{F}}$:

```
# let k' = fun x y -> y ;;
val k' : ['a] ['b > ['c] 'c -> 'c] 'a -> 'b
```

Here are graphs representing the type of K'. The left-hand graph represents its principal type in ML$^{\mathrm{F}}$, given above. The right-hand graph represents the type of K' in ML, that is, $\forall\,(\alpha,\beta)\ \alpha \to \beta \to \beta$.

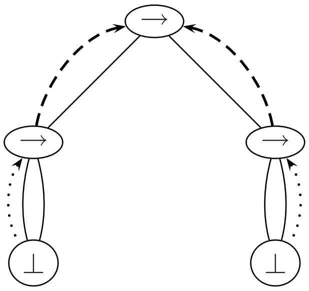


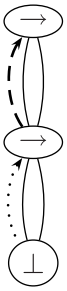


Example 5.2.10 This example emphasizes the role of the relation $\sqsupseteq$ during typing. Let f_1 and f_2 be two functions of respective types:

$$
\begin{array}{lcll}
\sigma_1 & \triangleq & \forall\,(\alpha = \sigma_{\mathtt{id}})\ \forall\,(\alpha' = \sigma_{\mathtt{id}}) & \alpha \to \alpha' \\
\sigma_2 & \triangleq & \forall\,(\alpha = \sigma_{\mathtt{id}}) & \alpha \to \alpha
\end{array}
$$

The graphs representing these types are the following:





Intuitively, f_1 and f_2 behave like the identity (they just return their argument). Let us check that the expression a_1 defined as $\lambda(x)\ (f_1\ x)\ x$, is typable but a_2, defined as $\lambda(x)\ (f_2\ x)\ x$, is not. In the former case, we can easily derive the following judgment, where Q stands for $(\alpha = \sigma_{\mathtt{id}}, \alpha' = \sigma_{\mathtt{id}})$ and Γ is $x : \alpha$:

$$\textsc{App}\ \frac{(Q)\ \Gamma \vdash f_1 : \alpha \to \alpha' \qquad (Q)\ \Gamma \vdash x : \alpha}{(Q)\ \Gamma \vdash f_1\ x : \alpha'\ (\underline{\mathbf{1}})}$$

Hence $(\alpha = \sigma_{\mathtt{id}})\ \Gamma \vdash f_1\ x : \forall\,(\alpha' = \sigma_{\mathtt{id}})\ \alpha'$ holds by Rule Gen since α' is not free in the context. By Rule Eq-Var, we get $\forall\,(\alpha' = \sigma_{\mathtt{id}})\ \alpha' \equiv \sigma_{\mathtt{id}}$ (under any prefix). Besides, we also have $\sigma_{\mathtt{id}} \sqsubseteq \alpha \to \alpha$ under any prefix that binds α. Consequently, we get $(\alpha = \sigma_{\mathtt{id}})\ \Gamma \vdash f_1\ x : \alpha \to \alpha$ by Rule Inst. Hence, $\vdash \lambda(x)\ (f_1\ x)\ x : \sigma_2$ follows by App and Gen. However, when f_1 is replaced by f_2, we can only derive $(Q)\ \Gamma \vdash f_2\ x : \alpha$ instead of the judgment (1) and we cannot apply Rule Gen with α. Additionally, it happens that $\sigma_1 \sqsubseteq \sigma_2$ is derivable (see for example Rule A-Alias'), thus $(f_2 : \star)$ can be given the type σ_1 so that $\lambda(x)\ (f_2 : \star)\ x\ x$ is typable.

We see that the expression a_2 is not typable as such, but becomes typable with an oracle. In this very example, the role of the oracle is to "transform" the function f_2 into f_1. The function f_2 can be viewed as the identity, that is, a silent function, whereas f_1 can be viewed as a type annotation. Hence, the oracle transforms a silent binding `let` $x = x$ `in` $x\ x$ into an annotated binding `let` $x = (x : \sigma_{\mathtt{id}})$ `in` $x\ x$.

Example 5.2.11 We recall that `app` is defined as $\lambda(x)\ \lambda(y)\ x\ y$. An interesting property of ML^{F} is that whenever an application $a_1\ a_2$ is typable, then `app` $a_1\ a_2$ is also typable, without any need for extra type annotations. Indeed, `app` has type $\forall\,(\alpha, \beta)\ (\alpha \to \beta) \to \alpha \to \beta$ (the typing derivation is similar to the one in ML). Besides, since $a_1\ a_2$ is typable, we must have a_1 of type $\tau_2 \to \tau_1$ and a_2 of type τ_2. Then by instantiation, `app` can be given the type $(\tau_2 \to \tau_1) \to \tau_2 \to \tau_1$, and `app` $a_1\ a_2$ is easily typed, then. It should be noted that, whereas only monotypes seem to be used, τ_1 and τ_2 can actually be type variables bound to polytypes in the current prefix. We see that although τ_1 and τ_2 can be aliases to polytypes, they are transparently propagated to the type of `app` by implicit instantiation.

Figure 5.3: Syntax-directed typing rules

$$\textsc{Var}^{\triangledown}\quad \frac{z : \sigma \in \Gamma}{(Q)\,\Gamma \vdash^{\triangledown} z : \sigma}$$

$$\textsc{Fun}^{\triangledown}\quad \frac{\begin{array}{c}\alpha \notin \mathsf{ftv}(\tau_0)\\ (QQ')\,\Gamma, x : \tau_0 \vdash^{\triangledown} a : \sigma \qquad \mathsf{dom}(Q') \mathrel{\#} \mathsf{ftv}(\Gamma)\end{array}}{(Q)\,\Gamma \vdash^{\triangledown} \lambda(x)\, a : \forall\,(Q', \alpha \geq \sigma)\, \tau_0 \to \alpha}$$

$$\textsc{App}^{\triangledown}\quad \frac{\begin{array}{c}(Q)\,\Gamma \vdash^{\triangledown} a_1 : \sigma_1 \qquad (Q)\,\Gamma \vdash^{\triangledown} a_2 : \sigma_2\\ (Q)\,\sigma_1 \sqsubseteq \forall\,(Q')\, \tau_2 \to \tau_1 \qquad (Q)\,\sigma_2 \sqsubseteq \forall\,(Q')\, \tau_2\end{array}}{(Q)\,\Gamma \vdash^{\triangledown} a_1\, a_2 : \forall\,(Q')\, \tau_1}$$

$$\textsc{Let}^{\triangledown}\quad \frac{\begin{array}{c}(Q)\,\Gamma \vdash^{\triangledown} a_1 : \sigma_1\\ (Q)\,\Gamma, x : \sigma_1 \vdash^{\triangledown} a_2 : \sigma_2\end{array}}{(Q)\,\Gamma \vdash^{\triangledown} \mathtt{let}\ x = a_1\ \mathtt{in}\ a_2 : \sigma_2}$$

$$\textsc{Oracle}^{\triangledown}\quad \frac{\begin{array}{c}(Q)\,\Gamma \vdash^{\triangledown} a : \sigma\\ (Q)\,\sigma \sqsubseteq \sigma'' \qquad (Q)\,\sigma'' \mathrel{\exists} \sigma'\end{array}}{(Q)\,\Gamma \vdash^{\triangledown} (a : \star) : \sigma'}$$

5.3 Syntax-directed presentation

As in ML, we can replace the typing rules of $\mathrm{ML}^{\mathrm{F}}_{\star}$ by a set of equivalent syntax-directed typing rules, which are given in Figure 5.3. Naively, a sequence of non-syntax-directed typing rules Gen and Inst should be placed around any other rule. However, many of these occurrences can be proved unnecessary by following an appropriate strategy. For instance, in ML, judgments are maintained instantiated as much as possible and are only generalized on the left-hand side of Rule Let. In $\mathrm{ML}^{\mathrm{F}}_{\star}$, this strategy would require more occurrences of generalization. Instead, we prefer to maintain typing judgments generalized as much as possible. Then it suffices to allow Rule Gen right after Rule Fun and to allow Rule Inst right before Rule App (see rules $\textsc{Fun}^{\triangledown}$ and $\textsc{App}^{\triangledown}$). Syntax-directed rules are slightly less intuitive. However, they are much more convenient for proving formal properties. In particular, induction on derivations can be replaced by structural induction on terms. The equivalence between the syntax-directed and original presentations of the typing rules is stated in section 6.2.

5.4 Dynamic semantics

The semantics of $\mathrm{ML}^{\mathrm{F}}_{\star}$ is the standard call-by-value semantics of ML. Another choice, such as *e.g.* call-by-name would satisfy the same essential properties, including type safety. However, the discussion about imperative features such as references, to be found in Part III, assumes a call-by-value semantics. We present it as a small-step reduction semantics. Values and call-by-value evaluation contexts are described below.

$$\begin{array}{rll} v ::= & w \mid (w : \star) & \\ w ::= & \lambda(x)\ a & \\ & \mid f\ v_1\ \ldots v_n & n < |f| \\ & \mid C\ v_1\ \ldots v_n & n \leq |C| \\ E ::= & [\,] \mid E\ a \mid v\ E \mid (E : \star) \mid \texttt{let}\ x = E\ \texttt{in}\ a & \end{array}$$

A value is either a function ($\lambda(x)\ a$), a partially applied primitive ($f\ v_1\ \ldots v_n$), or a constructed value $C\ v_1\ \ldots v_n$. Additionally, it can be an annotated value, that is $(w : \star)$, where w is a non-annotated value. We distinguish two classes, v and w, in order to forbid $((v : \star) : \star)$, which is not a value. Indeed, it can be reduced to $(v : \star)$ by Rule $\star\star$, to be found next. The reduction relation $\longrightarrow$ is parameterized by a set of δ-rules of the form (δ):

$$\begin{array}{rll} f\ v_1\ \ldots v_n \longrightarrow & a \quad \text{when } |f| = n & (\delta) \\ (\lambda(x)\ a)\ v \longrightarrow & a[v/x] & (\beta_v) \\ \texttt{let}\ x = v\ \texttt{in}\ a \longrightarrow & a[v/x] & (\beta_{let}) \\ (v_1 : \star)\ v_2 \longrightarrow & (v_1\ v_2 : \star) & (\star) \\ ((v : \star) : \star) \longrightarrow & (v : \star) & (\star\star) \end{array}$$

The main reduction is the β-reduction that takes two forms Rule (β_v) and Rule (β_{let}). Oracles are maintained during reduction, to which they do not contribute: they are simply pushed out of applications by Rule ($\star$). Moreover, two oracles do not have more power than one single oracle, as stated by Rule ($\star\star$). Finally, the reduction is the smallest relation containing (δ), (β_v), (β_{let}), ($\star$), and ($\star\star$) rules that is closed under E-congruence:

$$E[a] \longrightarrow E[a'] \text{ if } a \longrightarrow a' \qquad (\textsc{Context})$$

Note that the semantics of ML$^{\text{F}}$ is untyped. See Remark 9.3.1 on page 184 for more discussion of this issue.

Chapter 6

Type Safety

Type soundness of $\mathrm{ML}^{\mathrm{F}}_{\star}$ is shown as usual by a combination of *subject reduction*, which ensures that typings are preserved by reduction, and *progress*, which ensures that well-typed programs that are not values can be further reduced. A few standard results are shown first in Section 6.1. We show subject reduction and progress in the syntax-directed system; hence, we need to prove the equivalence between the syntax-directed system and the original one; this is done in Section 6.2. Type safety is shown for $\mathrm{ML}^{\mathrm{F}}_{\star}$ parameterized by a set of primitives and constructors. Of course, some assumptions about the behavior and the types of primitives are required. These assumptions, as well as subject reduction and progress, are addressed in Section 6.3.

6.1 Standard Properties

In this section, we give a few standard properties, which are similar to well known properties in ML. A first one states that typing judgments can be renamed. Then we show that the hypotheses gathered in the prefix can be instantiated (Rule WEAKEN) and that the hypotheses of the typing environment can be replaced by more general assumptions (Rule STRENGTHEN). Finally, the Substitutivity Lemma is a key result for showing subject-reduction.

6.1.1 Renaming and substitutions

Lemma 6.1.1 (Renaming of judgment) *Given a derivation of the judgment* (Q) $\Gamma \vdash a : \sigma$ *and a renaming* ϕ *on type variables, the judgment* $(\phi(Q))$ $\phi(\Gamma) \vdash a : \phi(\sigma)$ *admits a derivation of the same size.*

Proof: By induction on the derivation of $(Q)\ \Gamma \vdash a : \sigma$. All cases are easy. ■

6.1.2 Strengthening and weakening typing judgments

The instance relation between typing contexts $(Q)\ \Gamma' \sqsubseteq \Gamma$ is an abbreviation for

$$\forall z \in \mathsf{dom}(\Gamma),\ \text{we have } z \in \mathsf{dom}(\Gamma') \text{ and } (Q)\ \Gamma'(z) \sqsubseteq \Gamma(z)$$

The next lemma states two standard properties of typing judgments, presented in the form of rules Strengthen and Weaken, along with a variant Weaken*. In a judgment $(Q)\ \Gamma \vdash a : \sigma$, the polytype σ can be weakened as described by Rule Inst. Conversely, the context Γ can be strengthened, as described by Rule Strengthen. In addition to weakening the type or strengthening the context, the whole judgment can be instantiated. In ML, this is expressed by stability of typing judgments under substitutions. In ML$^{\mathsf{F}}$, this is modeled by instantiating the prefix of the typing judgment, as described by Rule Weaken. To see this, consider prefixes as a generalization of substitutions. An ML derivation $\Gamma \vdash a : \sigma$ would be represented in ML$^{\mathsf{F}}$ by $(Q)\ \Gamma \vdash a : \sigma$ where Q assigns unconstrained bounds to the type variables of $\mathsf{ftv}(\Gamma) \cup \mathsf{ftv}(\sigma)$. The substitution lemma says that for any substitution θ, the derivation $\theta(\Gamma) \vdash a : \theta(\sigma)$ $(\underline{1})$ is also valid. In ML$^{\mathsf{F}}$, applying Rule Weaken to $(Q)\ \Gamma \vdash a : \sigma$, we can first deduce that the derivation $(Q', (\alpha \geq \theta(\alpha))^{\alpha \in Q})\ \Gamma \vdash a : \sigma$ holds where Q' assigns unconstrained bounds to $\mathsf{ftv}(\theta(Q))$. By notation, this is $(Q', \underline{\theta})\ \Gamma \vdash a : \sigma$. Then we can derive $(Q')\ \theta(\Gamma) \vdash a : \theta(\sigma)$ (which represents the ML judgment (1)) using rules Strengthen, Gen and Inst and the equivalences $(Q'\underline{\theta})\ \Gamma \equiv \theta(\Gamma)$ and $(Q'\underline{\theta})\ \sigma \equiv \theta(\sigma)$. In summary, the stability of typing judgments under substitution in ML is here expressed as Rule Weaken, which allows the instantiation of the current prefix. Indeed, prefixes are a generalization of substitutions, and prefix instance corresponds to substitution composition. While Rule Weaken requires Q' to be an instance of Q under interface $\mathsf{dom}(Q)$, Rule Weaken* specifies an interface I which must only contain free variables of Γ and σ; this version is not used further.

Lemma 6.1.2 *The following rules are admissible:*

$$\textsc{Strengthen}\quad \frac{(Q)\ \Gamma \vdash a : \sigma \qquad (Q)\ \Gamma' \sqsubseteq \Gamma}{(Q)\ \Gamma' \vdash a : \sigma} \qquad\qquad \textsc{Weaken}\quad \frac{(Q)\ \Gamma \vdash a : \sigma \qquad Q \sqsubseteq Q'}{(Q')\ \Gamma \vdash a : \sigma}$$

$$\textsc{Weaken}^\star\quad \frac{(Q)\ \Gamma \vdash a : \sigma \qquad \mathsf{ftv}(\Gamma, \sigma) \subseteq I \qquad Q \sqsubseteq^I Q'}{(Q')\ \Gamma \vdash a : \sigma}$$

Proof: We prove STRENGTHEN by induction on the derivation of $(Q)\ \Gamma \vdash a : \sigma$. Cases APP, FUN, LET, INST and ORACLE are by induction hypothesis.

◦ CASE VAR: We have $(Q)\ \Gamma \vdash z : \sigma$ and $z : \sigma$ is in Γ. By definition, $z : \sigma'$ is in Γ', with $(Q)\ \sigma' \sqsubseteq \sigma$. Hence, we have (Rule VAR) $(Q)\ \Gamma' \vdash z : \sigma'$ and we get the expected result $(Q)\ \Gamma' \vdash z : \sigma$ by Rule INST.

◦ CASE GEN: The premises are $(Q, \alpha \diamond \sigma_a)\ \Gamma \vdash a : \sigma'$ (**1**) and $\alpha \notin \mathsf{ftv}(\Gamma)$. The conclusion is $(Q)\ \Gamma \vdash a : \forall\,(\alpha \diamond \sigma_a)\ \sigma'$. We have $(Q)\ \Gamma' \sqsubseteq \Gamma$ by hypothesis. By well-formedness all free type variables of Γ and Γ' must be bound in Q, and $\alpha \notin \mathsf{dom}(Q)$ holds by hypothesis, hence we have $\alpha \notin \mathsf{ftv}(\Gamma')$. The result is by induction hypothesis on (1) and Rule GEN, then.

We prove WEAKEN by induction on the derivation of $(Q)\ \Gamma \vdash a : \sigma$. Case VAR is immediate. Cases FUN, APP, and LET are by induction hypothesis. Case GEN: The premise is $(Q, \alpha \diamond \sigma_1)\ \Gamma \vdash a : \sigma_2$ (**2**) and $\alpha \notin \mathsf{ftv}(\Gamma)$. The conclusion is $(Q)\ \Gamma \vdash a : \forall\,(\alpha \diamond \sigma_1)\ \sigma_2$. By alpha-conversion, we can freely assume that $\alpha \notin \mathsf{dom}(Q')$. By Property 3.4.2.iii (page 106) and the hypothesis $Q \sqsubseteq Q'$, we get $(Q, \alpha \diamond \sigma_1) \sqsubseteq (Q', \alpha \diamond \sigma_1)$. By induction hypothesis and (2), we get a derivation of $(Q', \alpha \diamond \sigma_1)\ \Gamma \vdash a : \sigma_2$. By Rule GEN, we get $(Q')\ \Gamma \vdash a : \forall\,(\alpha \diamond \sigma_1)\ \sigma_2$, which is the expected result. Cases INST and ORACLE are direct consequences of Lemma 3.6.4.

Proof of WEAKEN*: The hypotheses are $(Q)\ \Gamma \vdash a : \sigma$ (**1**), $\mathsf{ftv}(\Gamma) \subseteq I$ (**2**), $\mathsf{ftv}(\sigma) \subseteq I$ (**3**), and $Q \sqsubseteq^I Q'$ (**4**). By Lemma 3.6.2 (page 114) and (4), there exists a renaming ϕ disjoint from I and a prefix Q_0 such that $Q \sqsubseteq \phi(Q')Q_0$ (**5**) and $\mathsf{dom}(Q_0) \# I$ (**6**) hold. Hence, by WEAKEN, (5) and (1), we have $(\phi(Q')Q_0)\ \Gamma \vdash a : \sigma$. By Rule GEN, (6) and (2), we get $(\phi(Q'))\ \Gamma \vdash a : \forall\,(Q_0)\ \sigma$ (**7**). By EQ-FREE, (6) and (3), we get $\forall\,(Q_0)\ \sigma \equiv \sigma$ (**8**). Hence, $(\phi(Q'))\ \Gamma \vdash a : \sigma$ (**9**) holds by Rule INST, (7), and (8). The renaming ϕ is disjoint from I, thus $\phi^\neg(\Gamma) = \Gamma$ and $\phi^\neg(\sigma) = \sigma$ hold by (2) and (3). By (9) and Lemma 6.1.1 applied with the renaming $\phi^\neg$, we get $(Q')\ \Gamma \vdash a : \sigma$, which is the expected result. ■

Corollary 6.1.3 *If QQ' is well-formed and if $(Q)\ \Gamma \vdash a : \sigma$ holds, then so does $(QQ')\ \Gamma \vdash a : \sigma$.*

As a consequence, the following rule is admissible:

$$\text{APP}^\star\quad \frac{(Q)\ \Gamma \vdash a_1 : \forall\,(Q')\ \tau_2 \to \tau_1 \qquad (Q)\ \Gamma \vdash a_2 : \forall\,(Q')\ \tau_2}{(Q)\ \Gamma \vdash a_1\ a_2 : \forall\,(Q')\ \tau_1}$$

Proof: Corollary 6.1.3 is a direct consequence of Rule WEAKEN, observing that $Q \equiv QQ'$ holds by PE-FREE.

App* is derivable: By alpha-conversion, we can assume that $Q' \# Q$ and $\mathsf{dom}(Q') \# \mathsf{ftv}(\Gamma)$. By Corollary 6.1.3, we have $(QQ')\ \Gamma \vdash a_1 : \forall (Q')\ \tau_2 \to \tau_1$ $\mathbf{(\underline{1})}$ as well as $(QQ')\ \Gamma \vdash a_2 : \forall (Q')\ \tau_2$ $\mathbf{(\underline{2})}$. By Rule I-Drop*, we have $(QQ')\ \forall (Q')\ \tau_2 \to \tau_1 \sqsubseteq \tau_2 \to \tau_1$ $\mathbf{(\underline{3})}$ as well as $(QQ')\ \forall (Q')\ \tau_2 \sqsubseteq \tau_2$ $\mathbf{(\underline{4})}$. Hence, $(QQ')\ \Gamma \vdash a_1 : \tau_2 \to \tau_1$ and $(QQ')\ \Gamma \vdash a_2 : \tau_2$ hold by Rule Inst, (1) and (3), as well as (2) and (4) respectively. Then $(QQ')\ \Gamma \vdash a_1\ a_2 : \tau_1$ holds by App. Finally, $(Q)\ \Gamma \vdash a_1\ a_2 : \forall (Q')\ \tau_1$ holds by Rule Gen. ■

6.1.3 Substitutivity

Lemma 6.1.4, next, is a usual result in ML, which states that unused hypotheses can be added to the typing environment.

Lemma 6.1.4 *If Γ and Γ' have disjoint domains, and if $(Q)\ \Gamma \vdash a : \sigma$ holds, then so does $(Q)\ \Gamma, \Gamma' \vdash a : \sigma$.*

See proof in the Appendix (page 289).

The substitutivity lemma is a key result for showing subject reduction.

Lemma 6.1.5 (Substitutivity) *If we have $(Q)\ \Gamma, x : \sigma \vdash a_0 : \sigma_0$ and $(Q)\ \Gamma \vdash a : \sigma$, then $(Q)\ \Gamma \vdash a_0[a/x] : \sigma_0$ holds.*

Proof: We write θ for the substitution $[a/x]$. By hypothesis, $(Q)\ \Gamma \vdash a : \sigma$ $\mathbf{(\underline{1})}$ holds. The proof is by induction on the derivation of $(Q)\ \Gamma, x : \sigma \vdash a_0 : \sigma_0$. Cases App, Inst, Gen and Oracle are by induction hypothesis.

∘ Case Fun: The premise is $(Q)\ \Gamma, x : \sigma, y : \tau_0 \vdash a_1 : \tau$ $\mathbf{(\underline{2})}$ and the conclusion is $(Q)\ \Gamma, x : \sigma \vdash \lambda(y)\ a_1 : \tau_0 \to \tau$. The judgment (2) can as well be written $(Q)\ \Gamma, y : \tau_0, x : \sigma \vdash a_1 : \tau$ $\mathbf{(\underline{3})}$. By Lemma 6.1.4 and (1), we have $(Q)\ \Gamma, y : \tau_0 \vdash a : \sigma$ $\mathbf{(\underline{4})}$. By induction hypothesis, (3), and (4) we have $(Q)\ \Gamma, y : \tau_0 \vdash \theta(a_1) : \tau$. Hence, by Rule Fun, we get $(Q)\ \Gamma \vdash \theta(\lambda(y)\ a_1) : \tau_0 \to \tau$, which is the expected result.

∘ Case Var: we have $(Q)\ \Gamma, x : \sigma \vdash z : \sigma_0$. If z is not x, then $\theta(z)$ is z, and we have $(Q)\ \Gamma \vdash \theta(z) : \sigma_0$. Otherwise, we have $z = x$ and $\sigma_0 = \sigma$. By hypothesis, $(Q)\ \Gamma \vdash x : \sigma$ holds, that is, $(Q)\ \Gamma \vdash a : \sigma$.

∘ Case Let: we have $(Q)\ \Gamma, x : \sigma \vdash \mathtt{let}\ y = a_1\ \mathtt{in}\ a_2 : \sigma_0$, and the premises are $(Q)\ \Gamma, x : \sigma \vdash a_1 : \sigma_y$ $\mathbf{(\underline{5})}$ and $(Q)\ \Gamma, x : \sigma, y : \sigma_y \vdash a_2 : \sigma_0$, which can also be written $(Q)\ \Gamma, y : \sigma_y, x : \sigma \vdash a_2 : \sigma_0$ $\mathbf{(\underline{6})}$. Thanks to Lemma 6.1.4 and (1), we have $(Q)\ \Gamma, y : \sigma_y \vdash a : \sigma$. By induction hypothesis applied once to (5) and once to (6), we get $(Q)\ \Gamma \vdash \theta(a_1) : \sigma_y$ and $(Q)\ \Gamma, y : \sigma_y \vdash \theta(a_2) : \sigma_0$. Thus $(Q)\ \Gamma \vdash \mathtt{let}\ y = \theta(a_1)\ \mathtt{in}\ \theta(a_2) : \sigma_0$ holds by Rule Let. This is equivalent to $(Q)\ \Gamma \vdash \theta(\mathtt{let}\ y = a_1\ \mathtt{in}\ a_2) : \sigma_0$, which is the expected result. ■

6.2 Equivalence between the syntax-directed system and the original system

The equivalence between the syntax-directed ($\vdash^{\triangledown}$) and original presentations ($\vdash$) of the typing rules is proved by showing the inclusion of $\vdash^{\triangledown}$ in $\vdash$ first. The inverse inclusion is not so direct, and we start by proving the equivalence between $\vdash$ and an intermediate type system ($\vdash'$), described below.

Lemma 6.2.1 *The system* $\vdash^{\triangledown}$ *is included in* $\vdash$*, that is, if* $(Q)\ \Gamma \vdash^{\triangledown} a : \sigma$ *is derivable, then* $(Q)\ \Gamma \vdash a : \sigma$ *is derivable.*

See proof in the Appendix (page 289).

We introduce an auxiliary system $\vdash'$ composed of the rules of $\vdash^{\triangledown}$ plus rules Gen and Inst.

Lemma 6.2.2 *The system* $\vdash$ *is included in* $\vdash'$*.*

<u>Proof</u>: Indeed, rules Var, Fun, App, Let, and Oracle can be derived using (respectively) Var$^{\triangledown}$, Fun$^{\triangledown}$ and Inst, App$^{\triangledown}$, Let$^{\triangledown}$, and Oracle$^{\triangledown}$. Hence, any derivation in the system $\vdash$ can be rewritten into a derivation into the system $\vdash'$. ■

Lemma 6.2.3 *The following rule is admissible:*

Strengthen'

$$\frac{(Q)\ \Gamma \vdash' a : \sigma \qquad (Q)\ \Gamma' \sqsubseteq \Gamma}{(Q)\ \Gamma' \vdash' a : \sigma}$$

The proof is similar to the proof of Rule Strengthen. The details can be found in the Appendix (page 290).

It remains to be shown that $\vdash'$ is included in $\vdash^{\triangledown}$. Intuitively, occurrences of Rule Gen can be moved up (and absorbed on the way up), whereas occurrences of Rule Inst can be moved down (and partially absorbed by Rule App).

Definition 6.2.4 The size of a typing derivation in the system $\vdash'$ is the number of rules it uses, ignoring the right premise in Rule Let$^{\triangledown}$. □

In the following properties, we show that the system $\vdash'$ keeps judgments as generalized as possible: Rule Gen can always be merged with other rules.

Properties 6.2.5

- ***i)*** *If we have a derivation of* $(Q)\ \Gamma \vdash' a : \sigma_a$ *ending with Rule* Gen*, then there exists* σ'_a *and a strictly smaller derivation of* $(Q)\ \Gamma \vdash' a : \sigma'_a$ *such that* $\sigma_a \equiv \sigma'_a$ *holds.*
- ***ii)*** *If we have a derivation of* $(Q)\ \Gamma \vdash' a : \sigma_a$*, then there exists* σ'_a *such that we have a derivation of* $(Q)\ \Gamma \vdash' a : \sigma'_a$ *not using* Gen*, and such that* $\sigma_a \equiv \sigma'_a$ *hold.*
- ***iii)*** *If we have a derivation of* $(Q)\ \Gamma \vdash' a : \sigma_a$ *not using* Gen*, then there exists* σ'_a *such that* $(Q)\ \Gamma \overset{\triangledown}{\vdash} a : \sigma'_a$ *and* $(Q)\ \sigma'_a \sqsubseteq \sigma_a$ *holds.*

See proof in the Appendix (page 290).

Thanks to these properties, it is now possible to state the equivalence between the syntax-directed system and the original system:

Lemma 6.2.6 (Syntax-Directed Typings) *We have* $(Q)\ \Gamma \vdash a : \sigma$ *iff there exists* σ' *such that* $(Q)\ \Gamma \overset{\triangledown}{\vdash} a : \sigma'$ *and* $(Q)\ \sigma' \sqsubseteq \sigma$*.*

Proof: If there exists σ' such that $(Q)\ \Gamma \overset{\triangledown}{\vdash} a : \sigma'$ and $(Q)\ \sigma' \sqsubseteq \sigma$ $(\underline{\mathbf{1}})$ hold, then $(Q)\ \Gamma \vdash a : \sigma'$ is derivable by Lemma 6.2.1, thus $(Q)\ \Gamma \vdash a : \sigma$ is derivable by Rule Inst and (1). Conversely, if $(Q)\ \Gamma \vdash a : \sigma$ holds, then there exists a derivation of $(Q)\ \Gamma \vdash' a : \sigma$ by Lemma 6.2.2. We get the expected result by Lemmas 6.2.5.ii and 6.2.5.iii, then. ■

Example 6.2.12 As we claimed in the introduction, a λ-bound variable that is used polymorphically must be annotated. Let us check that $\lambda(x)\ x\ x$ is not typable in ML^{F} by means of contradiction. A syntax-directed type derivation of this expression would be of the form:

$$\text{Fun}^{\triangledown}\ \dfrac{\text{App}^{\triangledown}\ \dfrac{\begin{array}{cc} \text{Var}^{\triangledown}\ (Q)\ x : \tau_0 \overset{\triangledown}{\vdash} x : \tau_0 & (Q)\ x : \tau_0 \overset{\triangledown}{\vdash} x : \tau_0\ \text{Var}^{\triangledown} \\ (Q)\ \tau_0 \sqsubseteq \forall (Q')\ \tau_2 \to \tau_1\ (\underline{\mathbf{2}}) & (Q)\ \tau_0 \sqsubseteq \forall (Q')\ \tau_2\ (\underline{\mathbf{1}}) \end{array}}{(Q)\ x : \tau_0 \overset{\triangledown}{\vdash} x\ x : \forall (Q')\ \tau_1}}{(Q)\ \emptyset \overset{\triangledown}{\vdash} \lambda(x)\ x\ x : \forall (\alpha \geq \forall (Q')\ \tau_1)\ \tau_0 \to \alpha}$$

By rules I-Drop*, (2), and (1), we get respectively $(QQ')\ \tau_0 \sqsubseteq \tau_2 \to \tau_1$ and $(QQ')\ \tau_0 \sqsubseteq \tau_2$. Then $(QQ')\ \tau_2 \to \tau_1 \equiv \tau_2$ follows by Lemma 2.1.6 and R-Trans. Thus, by Property 1.5.11.vii, there exists a substitution θ such that $\theta(\tau_2) = \theta(\tau_2 \to \tau_1)$, that is, $\theta(\tau_2) = \theta(\tau_2) \to \theta(\tau_1)$, which is impossible.

This example shows the limit of type inference, which is actually the strength of our system! We maintain principal types by rejecting examples where type inference would need to guess second-order types.

Example 6.2.13 Let us recover typability by introducing an oracle and building a derivation for $\lambda(x)\ (x : \star)\ x$. Taking $(\alpha = \sigma_{\mathtt{id}})$ for Q and α for τ_0, we obtain:

$$
\textsc{Fun}^{\triangledown}\ \dfrac{\textsc{App}^{\triangledown}\ \dfrac{\textsc{Oracle}^{\triangledown}\ \dfrac{\begin{array}{c}\textsc{Var}^{\triangledown}\ (Q)\ x : \alpha \vdash^{\triangledown} x : \alpha \\ (Q)\ \alpha \exists \sigma_{\mathtt{id}}\ (\underline{\mathbf{3}})\end{array}}{(Q)\ x : \alpha \vdash^{\triangledown} (x : \star) : \sigma_{\mathtt{id}}} \qquad (Q)\ x : \alpha \vdash^{\triangledown} x : \alpha\ \textsc{Var}^{\triangledown} \\ (Q)\ \sigma_{\mathtt{id}} \sqsubseteq \alpha \to \alpha}{(Q)\ x : \alpha \vdash^{\triangledown} (x : \star)\ x : \alpha}}{\vdash^{\triangledown} \lambda(x)\ (x : \star)\ x : \forall\,(\alpha = \sigma_{\mathtt{id}})\ \alpha \to \alpha}
$$

The oracle plays a crucial role in (3)—the revelation of the type scheme $\sigma_{\mathtt{id}}$ that is the bound of the type variable α used in the type of x. We have $(Q)\ \sigma_{\mathtt{id}} \sqsubseteq \alpha$, indeed, but the converse relation does not hold, so Rule Inst cannot be used here to replace α by its bound $\sigma_{\mathtt{id}}$. Chapter 8 shows how an explicit annotation $(x : \sigma_{\mathtt{id}})$ can replace the oracle $(x : \star)$.

6.3 Type safety

Of course, type soundness cannot hold without some assumptions relating the static semantics of constants described by the initial typing context Γ_0 and their dynamic semantics. To ease the presentation, we introduce a relation $\subseteq^{\Gamma}$ between programs for any typing environment Γ extending Γ_0: we write $a \subseteq^{\Gamma} a'$ if and only if every typing of a under Γ, *i.e.* every pair (Q, σ) such that $(Q)\ \Gamma \vdash a : \sigma$ holds, is also a typing of a'. A relation $\mathcal{R}$ on programs preserves typings under Γ whenever it is a sub-relation of $\subseteq^{\Gamma}$. Then showing subject-reduction for a given reduction relation $\mathcal{R}$ under a typing environment Γ amounts to showing that $\mathcal{R}$ preserve typings under Γ.

We now introduce the three hypotheses that the constants are required to validate.

Definition 6.3.1 (Hypotheses) We assume that the following properties hold for constants.

(H0) (Arity) Each constant $c \in \mathsf{dom}(\Gamma_0)$ has a closed type $\Gamma_0(c)$ of the form $\forall\,(Q)\ \tau_1 \to \ldots \tau_{|c|} \to \tau$ and such that $(\forall\,(Q)\ \tau)/\epsilon$ is not in $\{\to, \bot\}$ whenever c is a constructor.

(H1) (Subject-Reduction) All δ-rules preserve typings. That is (δ) is a sub-relation of $\subseteq^{\Gamma}$, for any Γ extending Γ_0.

(H2) (Progress) Any expression a of the form $f\ v_1 \ldots v_{|f|}$, such that $(Q)\ \Gamma \vdash a : \sigma$ is in the domain of (δ). □

The first hypothesis (H0) links the dynamic arity $|c|$ of a constant c to its type. Intuitively, a constant of arity n must have a type with exactly n arrows, no more. The second hypothesis (H1) simply states subject-reduction for each constant. The third hypothesis (H2) is progress for primitives, that is, when a primitive f is applied to $|f|$ arguments, it must reduce to a value.

The language $\text{ML}^{\text{F}}_{\star}$ is parameterized by a set of primitives and constants that satisfy the hypotheses above. These hypotheses are sufficient to ensure type safety, which we prove by a combination of subject-reduction, next, and progress. In ML^{F} and $\text{ML}^{\text{F}}_{\star}$, subject reduction holds only under Γ_0 (actually, under any closed typing environment extending Γ_0). This means that reduction is only allowed at top-level, and not under a λ-abstraction. As a counterexample, consider the expression $\lambda(x)\ (x : \star)\ x$. It is typable in $\text{ML}^{\text{F}}_{\star}$. However, if we allowed reduction under λ-abstractions, we could use Rule ($\star$) to reduce the above expression to $\lambda(x)\ (x\ x : \star)$, which is not typable in $\text{ML}^{\text{F}}_{\star}$. It is typable in UML^{F}, though, adding an implicit oracle on the first occurrence of x. We see that subject reduction does not hold under any context in $\text{ML}^{\text{F}}_{\star}$. This is why subject reduction is stated only under under the initial typing environment Γ_0 (which may contain any closed bindings, though). Conversely, subject reduction does hold under any typing environment in UML^{F}. This means that partial reduction is not possible in ML^{F} because it would lose type information needed for type inference. However, it is possible in UML^{F} where oracles are implicit. We formalize this by giving two statements for subject reduction.

Theorem 2 (Subject reduction) *Reduction preserves typings in UML^{F}, and preserve typings under Γ_0 in $\text{ML}^{\text{F}}_{\star}$.*

Proof: We must show that $(\longrightarrow)$ is a subrelation of $(\sqsubseteq^{\Gamma})$ in UML^{F} and a subrelation of $(\sqsubseteq^{\Gamma_0})$ in $\text{ML}^{\text{F}}_{\star}$. We proceed by case analysis on the different reduction rules. In each case, by hypothesis, we have an expression a_1 that reduces to a_2 and a derivation of $(Q)\ \Gamma \vdash a_1 : \sigma$ (**1**). We must show that $(Q)\ \Gamma \vdash a_2 : \sigma$ holds (**2**). By Lemma 6.2.6 and (1), we have a derivation in the syntax-directed system, that is, we have $(Q)\ \Gamma \overset{\triangledown}{\vdash} a_1 : \sigma'$ (**3**) with $(Q)\ \sigma' \sqsubseteq \sigma$ (**4**). It suffices to show that $(Q)\ \Gamma \vdash a_2 : \sigma'$ (**5**) holds. Indeed, (2) then holds by Inst and (4). We show either (2) or (5), depending which one is easier. We note that the rules ($\star$) and ($\star\star$) make no sense in UML^{F} (indeed, since oracles can be implicitly inserted, these reduction rules have no effect). Hence, we consider the cases ($\star$) and ($\star\star$) only in $\text{ML}^{\text{F}}_{\star}$, which imply that the typing environment is Γ_0.

○ Case δ: The judgment (2) holds by hypothesis (H1).

○ Case β_{let}: The reduction is $\texttt{let}\ x = v\ \texttt{in}\ a \longrightarrow a[v/x]$, and we know by (3) that the judgment $(Q)\ \Gamma \overset{\triangledown}{\vdash} \texttt{let}\ x = v\ \texttt{in}\ a : \sigma'$ holds. The derivation must end as follows:

$$\frac{(Q)\ \Gamma \overset{\triangledown}{\vdash} v : \sigma_1 \qquad (Q)\ \Gamma, x : \sigma_1 \overset{\triangledown}{\vdash} a : \sigma'}{(Q)\ \Gamma \overset{\triangledown}{\vdash} \texttt{let}\ x = v\ \texttt{in}\ a : \sigma'}\ \text{Let}^{\triangledown}$$

Applying the Substitutivity Lemma (Lemma 6.1.5), we get $(Q)\ \Gamma \vdash^{\triangledown} a[v/x] : \sigma'$, which is the expected result (5).

∘ Case β_v: The reduction is $(\lambda(x)\ a)\ v \longrightarrow a[v/x]$, and we know that the judgment $(Q)\ \Gamma \vdash^{\triangledown} (\lambda(x)\ a)\ v : \sigma'$ holds. By definition of $\vdash^{\triangledown}$, there must exist Q_1, Q_2, τ, τ_1, τ_2, σ_1, σ_2, and α such that this derivation is of the form:

$$\dfrac{\begin{array}{c} \textsc{Fun}^{\triangledown}\ \dfrac{(QQ_1)\ \Gamma, x : \tau \vdash^{\triangledown} a : \sigma_1\ \ (\underline{\mathbf{9}}) \qquad \mathsf{dom}(Q_1) \cap \Gamma = \emptyset}{(Q)\ \Gamma \vdash^{\triangledown} \lambda(x)\ a : \forall\,(Q_1, \alpha \geq \sigma_1)\ \tau \to \alpha} \qquad (Q)\ \Gamma \vdash^{\triangledown} v : \sigma_2\ \ (\underline{\mathbf{8}}) \\ (Q)\ \forall\,(Q_1, \alpha \geq \sigma_1)\ \tau \to \alpha \sqsubseteq \forall\,(Q_2)\ \tau_2 \to \tau_1\ \ (\underline{\mathbf{7}}) \qquad (Q)\ \sigma_2 \sqsubseteq \forall\,(Q_2)\ \tau_2\ \ (\underline{\mathbf{6}}). \end{array}}{(Q)\ \Gamma \vdash^{\triangledown} (\lambda(x)\ a)\ v : \forall\,(Q_2)\ \tau_1}\ \textsc{App}^{\triangledown}$$

Here σ' is $\forall\,(Q_2)\ \tau_1$. Let Q'_1 be the prefix $(Q_1, \alpha \geq \sigma_1)$. Without lost of generality, we can assume $\alpha \notin \mathsf{dom}(Q)$ (otherwise, we rename α). This implies $Q'_1 \# Q$; besides $\mathsf{dom}(Q'_1) \# \mathsf{ftv}(\Gamma)$ $(\underline{\mathbf{10}})$ since Γ must be closed under Q by well-formedness of (8). We can also assume $\mathsf{dom}(Q'_1) = \mathsf{dom}(Q'_1/\tau \to \alpha)$ $(\underline{\mathbf{11}})$ (otherwise Q'_1 contains useless binders that can be freely removed). We get a derivation of $(QQ'_1)\ \Gamma, x : \tau \vdash a : \alpha$ $(\underline{\mathbf{12}})$ by Corollary 6.1.3, Inst and I-Hyp applied to (9). From (7), we get $(Q)\ \forall\,(Q'_1)\ \tau \to \alpha \sqsubseteq \forall\,(Q_2)\ \tau_2 \to \tau_1$ $(\underline{\mathbf{13}})$. Applying Lemma 3.4.4 to (13), there exists a substitution θ and an alpha-conversion of $\forall\,(Q_2)\ \tau_2 \to \tau_1$, written $\forall\,(Q'_2)\ \tau'_2 \to \tau'_1$ such that $(QQ'_2)\ \theta(\tau \to \alpha) \equiv \tau'_2 \to \tau'_1$ $(\underline{\mathbf{14}})$ and $QQ'_1 \sqsubseteq^I QQ'_2\underline{\theta}$ $(\underline{\mathbf{15}})$, where $I = \mathsf{dom}(Q) \cup \mathsf{dom}(Q'_1/\tau \to \alpha)$ and $\mathsf{dom}(\theta) \subseteq \mathsf{dom}(Q'_1/\tau \to \alpha)$ $(\underline{\mathbf{16}})$. Note that $\mathsf{ftv}(\Gamma) \subseteq \mathsf{dom}(Q)$ $(\underline{\mathbf{17}})$ hold by well-formedness of (8). From (15) and (11), we get $QQ'_1 \sqsubseteq QQ'_2\underline{\theta}$ $(\underline{\mathbf{18}})$. Hence, by Rule Weaken, (12), and (18), we get $(QQ'_2\underline{\theta})\ \Gamma, x : \tau \vdash a : \alpha$. By Strengthen, and observing that $(QQ'_2\underline{\theta})\ \sigma_0 \equiv \theta(\sigma_0)$ holds by Eq-Mono for any σ_0, we get the judgment $(QQ'_2\underline{\theta})\ \theta(\Gamma), x : \theta(\tau) \vdash a : \alpha$. By Gen, we get $(QQ'_2)\ \theta(\Gamma), x : \theta(\tau) \vdash a : \forall\,(\underline{\theta})\ \alpha$. By Inst and $(QQ'_2)\ \forall\,(\underline{\theta})\ \alpha \equiv \theta(\alpha)$, it leads to $(QQ'_2)\ \theta(\Gamma), x : \theta(\tau) \vdash a : \theta(\alpha)$ $(\underline{\mathbf{19}})$. From (14) and Property 1.5.11.viii (page 54), we get $(QQ'_2)\ \theta(\tau) \equiv \tau'_2$ and $(QQ'_2)\ \theta(\alpha) \equiv \tau'_1$. Hence, by Strengthen and Inst on (19), we get $(QQ'_2)\ \theta(\Gamma), x : \tau'_2 \vdash a : \tau'_1$ $(\underline{\mathbf{20}})$. By (16) and (10), we get $\mathsf{dom}(\theta) \# \mathsf{dom}(Q)$. Hence, by (17), we get $\theta(\Gamma) = \Gamma$. Then, from (20) we have the judgment $(QQ'_2)\ \Gamma, x : \tau'_2 \vdash a : \tau'_1$ $(\underline{\mathbf{21}})$. From (6), we get $(Q)\ \sigma_2 \sqsubseteq \forall\,(Q'_2)\ \tau'_2$ by alpha-conversion. Hence, $(QQ'_2)\ \sigma_2 \sqsubseteq \tau'_2$ $(\underline{\mathbf{22}})$ holds by Property 1.7.2.iii (page 59) and I-Drop*. From (8) and Corollary 6.1.3 (page 153), we get $(QQ'_2)\ \Gamma \vdash v : \sigma_2$. By Inst and (22), we get $(QQ'_2)\ \Gamma \vdash v : \tau'_2$ $(\underline{\mathbf{23}})$. Hence, applying the Substitutivity Lemma (Lemma 6.1.5) to (21) and (23), we get $(QQ'_2)\ \Gamma \vdash a[v/x] : \tau'_1$ $(\underline{\mathbf{24}})$. Moreover, we have $\mathsf{dom}(Q'_2) \# \mathsf{ftv}(\Gamma)$ from (17), so that applying Rule Gen to (24) leads to $(Q)\ \Gamma \vdash a[v/x] : \forall\,(Q'_2)\ \tau'_1$, that is, by alpha-conversion $(Q)\ \Gamma \vdash a[v/x] : \forall\,(Q_2)\ \tau_1$. This is the expected result.

∘ Case $(\star)$: The reduction is $(v_1 : \star)\ v_2 \longrightarrow (v_1\ v_2 : \star)$ and a is $(v_1 : \star)\ v_2$. As explained above, this case concerns only $\mathrm{ML}^{\mathrm{F}}_{\star}$; it cannot occur in $\mathrm{UML}^{\mathrm{F}}$. As a consequence, the typing environment we consider is the initial typing environment Γ_0. By hypothesis (H0), Γ_0 is closed, that is, $\mathsf{ftv}(\Gamma_0) = \emptyset$. Applying repeatedly Rule Gen to (1), we get $(\emptyset)\ \Gamma_0 \vdash (v_1 : \star)\ v_2 : \forall\,(Q)\ \sigma$. By Lemma 6.2.6, we have a derivation of $(\emptyset)\ \Gamma_0 \vdash^{\triangledown} (v_1 : \star)\ v_2 : \forall\,(Q')\ \tau_1$ such that $\forall\,(Q')\ \tau_1 \sqsubseteq \forall\,(Q)\ \sigma$ $(\underline{\mathbf{25}})$. This derivation necessarily ends with the following rules:

$$\textsc{App}^{\triangledown}\;\dfrac{\begin{array}{cc}\textsc{Oracle}^{\triangledown}\;\dfrac{\begin{array}{c}(\emptyset)\,\Gamma_0 \vdash^{\triangledown} v_1 : \sigma_1 \quad (\underline{\mathbf{31}})\\ \sigma_1 \sqsubseteq \sigma_1'' \ (\underline{\mathbf{30}}) \qquad \sigma_1' \mathrel{-\!\!\!\sqsubseteq} \sigma_1'' \ (\underline{\mathbf{29}})\end{array}}{(\emptyset)\,\Gamma_0 \vdash^{\triangledown} (v_1 : \star) : \sigma_1'} & (\emptyset)\,\Gamma_0 \vdash^{\triangledown} v_2 : \sigma_2 \quad (\underline{\mathbf{28}})\\ \sigma_1' \sqsubseteq \forall\,(Q')\ \tau_2 \to \tau_1 \quad (\underline{\mathbf{27}}) & \sigma_2 \sqsubseteq \forall\,(Q')\ \tau_2 \quad (\underline{\mathbf{26}})\end{array}}{(\emptyset)\,\Gamma_0 \vdash^{\triangledown} (v_1 : \star)\ v_2 : \forall\,(Q')\ \tau_1}$$

The relations between types, above, can be represented by the following diagram:

$$\begin{tikzcd}
 & \sigma_1' \arrow[r, "\sqsubseteq^{(27)}"] \arrow[d, "\mathrel{-\!\!\!\sqsubseteq}^{(29)}"] & \forall\,(Q')\ \tau_2 \to \tau_1 \\
\sigma_1 \arrow[r, "\sqsubseteq^{(30)}"] & \sigma_1'' &
\end{tikzcd}$$

Applying the Diamond Lemma (Lemma 2.8.4) to the above diagram, we know that there exists σ_3 such that the following holds:

$$\begin{tikzcd}
 & \sigma_1' \arrow[r, "\sqsubseteq^{(27)}"] \arrow[d, "\mathrel{-\!\!\!\sqsubseteq}"] & \forall\,(Q')\ \tau_2 \to \tau_1 \arrow[d, dotted, "\mathrel{-\!\!\!\sqsubseteq}^{(\underline{\mathbf{32}})}"] \\
\sigma_1 \arrow[r, "\sqsubseteq^{(30)}"] & \sigma_1'' \arrow[r, dotted, "\sqsubseteq^{(\underline{\mathbf{33}})}"] & \sigma_3
\end{tikzcd}$$

We can freely assume that σ_3 is in normal form. By Property 2.1.3.ii (page 65) and (32), we conclude that σ_3 is of the form $\forall\,(Q_3)\ \tau_2' \to \tau_1'$. By Lemma 3.6.11 and (32), $\forall\,(Q')\ \tau_2 \mathrel{-\!\!\!\sqsubseteq} \forall\,(Q_3)\ \tau_2'$ **(<u>34</u>)** and $\forall\,(Q')\ \tau_1 \mathrel{-\!\!\!\sqsubseteq} \forall\,(Q_3)\ \tau_1'$ **(<u>35</u>)** hold. By Rule Inst, (31), (30), and (33) we get $(\emptyset)\,\Gamma_0 \vdash v_1 : \forall\,(Q_3)\ \tau_2' \to \tau_1'$ **(<u>36</u>)**. By Rule Inst, (28), (26), and (34), we can derive $(\emptyset)\,\Gamma_0 \vdash v_2 : \forall\,(Q_3)\ \tau_2'$ **(<u>37</u>)**. Consequently, by Rule App*, (36) and (37), $(\emptyset)\,\Gamma_0 \vdash v_1\ v_2 : \forall\,(Q_3)\ \tau_1'$ holds. Finally, by Oracle and (35), we get $(\emptyset)\,\Gamma_0 \vdash (v_1\ v_2 : \star) : \forall\,(Q')\ \tau_1$. By (25) and Inst, we get $(\emptyset)\,\Gamma_0 \vdash (v_1\ v_2 : \star) : \forall\,(Q)\ \sigma$, that is, $(\emptyset)\,\Gamma_0 \vdash a' : \forall\,(Q)\ \sigma$. By Corollary 6.1.3 (page 153), we get $(Q)\,\Gamma_0 \vdash a' : \forall\,(Q)\ \sigma$. Besides, $(Q)\ \forall\,(Q)\ \sigma \sqsubseteq \sigma$ holds by I-Drop*. Hence, $(Q)\,\Gamma_0 \vdash a' : \sigma$ holds by Inst. This is the expected result (2).

○ Case ($\star\star$): This case concerns only $\mathrm{ML}^{\mathrm{F}}_{\star}$, thus the typing environment is Γ_0. We have a derivation of $(Q)\,\Gamma_0 \vdash ((a_0 : \star) : \star) : \sigma$. As in the previous case ($\star$), we have $(\emptyset)\,\Gamma_0 \vdash ((a_0 : \star) : \star) : \forall\,(Q)\ \sigma$ by Gen. By Lemma 6.2.6, we have a derivation of $(\emptyset)\,\Gamma_0 \vdash^{\triangledown} ((a_0 : \star) : \star) : \sigma_0$ **(<u>38</u>)**, with $\sigma_0 \sqsubseteq \forall\,(Q)\ \sigma$ **(<u>39</u>)**. The derivation of (38) is necessarily of the form

$$\textsc{Oracle}^{\triangledown}\;\dfrac{\textsc{Oracle}^{\triangledown}\;\dfrac{(\emptyset)\,\Gamma_0 \vdash^{\triangledown} a_0 : \sigma_3 \quad (\underline{\mathbf{40}}) \qquad (\emptyset)\ \sigma_3 \sqsubseteq \sigma_4 \qquad (\emptyset)\ \sigma_1 \mathrel{-\!\!\!\sqsubseteq} \sigma_4}{(\emptyset)\,\Gamma_0 \vdash^{\triangledown} (a_0 : \star) : \sigma_1} \qquad (\emptyset)\ \sigma_1 \sqsubseteq \sigma_2 \qquad (\emptyset)\ \sigma_0 \mathrel{-\!\!\!\sqsubseteq} \sigma_2}{(\emptyset)\,\Gamma_0 \vdash^{\triangledown} ((a_0 : \star) : \star) : \sigma_0}$$

We can represent the relation between these polytypes by the solid arrows in the following diagram:

$$\begin{tikzcd}
 & & \sigma_0 \arrow[d, "\mathrel{\sqsubset\!\!\!\!-}^{(\underline{\mathbf{42}})}"] \\
 & \sigma_1 \arrow[r, "\sqsubseteq"] \arrow[d, "\mathrel{\sqsubset\!\!\!\!-}"] & \sigma_2 \arrow[d, dotted, "\mathrel{\sqsubset\!\!\!\!-}^{(\underline{\mathbf{44}})}"] \\
\sigma_3 \arrow[r, "\sqsubseteq^{(\underline{\mathbf{41}})}"] & \sigma_4 \arrow[r, dotted, "\sqsubseteq^{(\underline{\mathbf{43}})}"] & \sigma_5
\end{tikzcd}$$

By the Diamond Lemma 2.8.4, there exists σ_5 such that the relations represented by the dotted arrows hold. By R-Trans, (41), and (43), we get $\sigma_3 \sqsubseteq \sigma_5$ **(<u>45</u>)**. By R-Trans, (42), and (44), we get $\sigma_0 \mathrel{\sqsubset\!\!\!\!-} \sigma_5$ **(<u>46</u>)**. Hence, by (40), Oracle$^{\triangledown}$, (45), and (46), we get $(\emptyset)\ \Gamma_0 \overset{\triangledown}{\vdash} (a_0 : \star) : \sigma_0$. By Inst and (39), we get $(\emptyset)\ \Gamma_0 \vdash (a_0 : \star) : \forall (Q)\ \sigma$. By Corollary 6.1.3 (page 153), Inst and I-Drop*, we can derive $(Q)\ \Gamma_0 \vdash (a_0 : \star) : \sigma$. This is the expected result (2).

○ Case Context: The hypothesis is $a \longrightarrow a'$. We know that $(\emptyset)\ \Gamma \vdash E[a] : \sigma'$ holds. We must show that $(Q)\ \Gamma \vdash E[a'] : \sigma$ (2) also holds. The proof is immediate by structural induction on E: ■

The second step in showing type safety consists of showing progress.

Theorem 3 (Progress) *Any expression a such that we have $(Q)\ \Gamma_0 \vdash a : \sigma$ is a value or can be further reduced.*

<u>Proof</u>: We reason by induction on the structure of a.

○ Case x: This is not possible, since Γ_0 does not bind any variable.

○ Case c: Then a is a value.

○ Case $\lambda(x)\ a$: Then a is a value.

○ Case `let` $x = a_1$ `in` a_2: Necessarily, a_1 is typable in the typing environment Γ_0. By induction hypothesis, either a_1 is a value or it can be further reduced. In either case, a can be further reduced (by Rule β_{let} or by Rule Context).

○ Case $a_1\ a_2$ Necessarily, a_1 and a_2 are also well-typed in $(Q), \Gamma_0$. Thus, by induction hypothesis, either a_1 can be further reduced (and so can a) or a_1 is a value. We continue with the second case. In turn, either a_2 can be further reduced and so can a or it is also a value. We continue with the second case and reason on the structure of a_1:

Subcase $(w : \star)$: then a can be reduced by $(\star)$.

Subcase $\lambda(x)\ a'$: then a can be reduced by β_v.

Subcase $f\ v_1\ \ldots v_n$ with $n < |f|$: then either $n+1 < |f|$ and a is a value, or $n+1 = |f|$ and a can be reduced by hypothesis **(H2)** for constants.

Subcase $C\ v_1\ \ldots v_n$ with $n \leq |C|$: By Lemma 6.2.6, we have a typing derivation of $(Q)\ \Gamma_0 \overset{\triangledown}{\vdash} a_1 : \sigma$. This derivation starts with Var$^\triangledown$ and uses App$^\triangledown$ repeatedly, once for each v_i. Let σ_i be the type given to $C\ v_1\ \ldots v_i$ in the i^{th} Rule App$^\triangledown$. Let σ_0 be $\Gamma(C)$. We show by induction on the number of rules App$^\triangledown$, that $(\forall\,(Q)\ \sigma_i)/2^{|C|-i}$ is neither $\rightarrow$, nor $\bot$.

- This result holds for $i = 0$ by hypothesis (H0).
- Assuming the result holds for i, we show it also holds for $i+1$: By Rule App$^\triangledown$, we have $(Q)\ \sigma_i \sqsubseteq \forall\,(Q')\ \tau_2 \rightarrow \tau_1$ and σ_{i+1} is $\forall\,(Q')\ \tau_1$. By Property 2.1.3.ii (page 65), $(\forall\,(QQ')\ \tau_2 \rightarrow \tau_1)/2^{|C|-i}$ is neither $\rightarrow$, nor $\bot$. Hence, $(\forall\,(QQ')\ \tau_1)/2^{|C|-(i+1)}$ is neither $\rightarrow$, nor $\bot$, which is the expected result.

By induction, the result holds for all $i \in 0..|C|$. If n is $|C|$, then $(\forall\,(Q)\ \sigma_n)/\epsilon$ is neither $\rightarrow$ nor $\bot$ **(1)**. Besides, Rule App$^\triangledown$ is used to type $a_1\ a_2$, which implies that $(Q)\ \sigma_n \sqsubseteq \forall\,(Q')\ \tau_2 \rightarrow \tau_1$ **(2)** holds for some Q', τ_2 and τ_1. We have a contradiction between (1), (2), and Property 2.1.3.ii. Hence n cannot be $|C|$. Thus, a is a value.

○ Case $(a' : \star)$: Necessarily, a' is well-typed. By induction hypothesis, either a' is a value or it can be further reduced. In the latter case, a can be reduced by Rule Context. In the former case, a' is either a value of the form w, and a is a value, or it is a value v of the form $(w : \star)$. Then a can be reduced by $(\star\star)$. ■

Combining theorems 2 and 3 ensures that the reduction of well-typed programs either proceeds forever or ends up with a value. This holds for programs in $\text{ML}^{\text{F}}_\star$ but also for programs in ML^{F}, since ML^{F} is a subset of $\text{ML}^{\text{F}}_\star$. Hence ML^{F} is also sound. However, ML^{F} does not enjoy subject reduction, since reduction may create oracles. Notice, however, that oracles can only be introduced by δ-rules.

Chapter 7

Type inference

Type inference in $\mathrm{ML}^{\mathrm{F}}_{\star}$ is probably as difficult as full type inference in System F, because oracles do not provide any type information. Indeed, oracles only indicate place-holders for type annotations. Fortunately, a program in ML^{F} mentions type annotations, which are primitives, instead of oracles. In this chapter, we give a type-inference algorithm for ML^{F}, which we show sound and complete. Interestingly, type annotations are not considered here, because type annotations are primitives, and the type of primitives is always known (from the initial typing environment Γ_0). Hence, the type inference algorithm does not guess any polymorphism, but simply propagates known type information. This is why the algorithm is quite simple, and also why it is complete.

7.1 Type inference algorithm

Figure 7.1 defines the type-inference algorithm W^{F} for ML^{F}. The algorithm follows the algorithm W for ML [Mil78], with only two differences: first, the algorithm builds a prefix Q instead of a substitution; second, all free type variables not in Γ are quantified at each abstraction or application. Since free variables of Γ are in $\mathsf{dom}(Q)$, finding quantified variables consists of splitting the current prefix according to $\mathsf{dom}(Q)$, as described by Definition 3.5.1.

Definition 7.1.1 A pair (Q', σ') is an *instance* of a pair (Q, σ) under interface I if we have $(Q, \alpha \geq \sigma) \sqsubseteq^{I \cup \{\alpha\}} (Q', \alpha \geq \sigma')$ for any α not in $\mathsf{dom}(Q) \cup \mathsf{dom}(Q')$. □

A *type inference problem* is a triple (Q, Γ, a), where all free type variables in Γ are bound in Q. A pair (Q', σ) is a *solution* to this problem if $Q \sqsubseteq Q'$ and $(Q')\ \Gamma \vdash a : \sigma$ holds. A solution of a type inference problem (Q, Γ, a) is *principal* if all other solutions are instances of the given one, under interface $\mathsf{dom}(Q)$.

Figure 7.1: Algorithm W^{F}

```
The algorithm infer $(Q, \Gamma, a)$ is defined by cases on expression $a$ :
Case $x$ : return $Q, \Gamma(x)$
Case $\lambda(x)\ a$ :
    • let $Q_1 = (Q, \alpha \geq \bot)$ with $\alpha \notin \mathsf{dom}(Q)$
    • let $(Q_2, \sigma)$ = infer $(Q_1, (\Gamma, x : \alpha), a)$
    • let $\beta \notin \mathsf{dom}(Q_2)$ and $(Q_3, Q_4) = Q_2 \upharpoonright \mathsf{dom}(Q)$
    • return $Q_3, \forall\, (Q_4, \beta \geq \sigma)\ \alpha \to \beta$
Case $a\ b$ :
    • let $(Q_1, \sigma_a)$ = infer $(Q, \Gamma, a)$
    • let $(Q_2, \sigma_b)$ = infer $(Q_1, \Gamma, b)$
    • let $\alpha_a, \alpha_b, \beta \notin \mathsf{dom}(Q_2)$
    • let $Q_3$ = unify $((Q_2, \alpha_a \geq \sigma_a, \alpha_b \geq \sigma_b, \beta \geq \bot), \alpha_a, \alpha_b \to \beta)$
    • let $(Q_4, Q_5) = Q_3 \upharpoonright \mathsf{dom}(Q)$
    • return $(Q_4, \forall\, (Q_5)\ \beta)$
Case let $x = a_1$ in $a_2$ :
    • let $(Q_1, \sigma_1)$ = infer $(Q, \Gamma, a_1)$
    • return infer $(Q_1, (\Gamma, x : \sigma_1), a_2)$
```

7.2 Soundness of the algorithm

Lemma 7.2.1 (Soundness of type inference) *The algorithm W^F is sound; that is, for any Q, Γ, and a, if* `infer` (Q, Γ, a) *returns a pair (Q', σ), then we have:*

$$Q \sqsubseteq Q' \qquad\qquad (Q')\ \Gamma \vdash a : \sigma$$

Proof: By structural induction on a.

∘ CASE x: We have $Q' = Q$ and $\sigma = \Gamma(x)$, thus the result is immediate by Rule VAR.

∘ CASE $\lambda(x)\ a$: Let Q_1 be $(Q, \alpha \geq \bot)$ and (Q_2, σ) be `infer` $(Q_1, \Gamma, x : \alpha, a)$. By induction hypothesis, we have $Q_1 \sqsubseteq Q_2$ and $(Q_2)\ \Gamma, x : \alpha \vdash a : \sigma$ (**1**). Let (Q_3, Q_4) be $Q_2 \upharpoonright \mathsf{dom}(Q)$. We have $Q_2 \equiv Q_3 Q_4$ (**2**) by Lemma 3.5.2. Hence, by WEAKEN, (2), and (1), we have $(Q_3 Q_4)\ \Gamma, x : \alpha \vdash a : \sigma$. By Rule FUN$^{\triangledown}$, we get $(Q_3)\ \Gamma \vdash \lambda(x)\ a : \forall\, (Q_4, \beta \geq \sigma)\ \alpha \to \beta$, which is the expected result. Moreover, we have $Q \equiv Q_1 \sqsubseteq Q_2 \equiv Q_3 Q_4 \equiv^{\mathsf{dom}(Q_1)} Q_3$. Hence $Q \sqsubseteq Q_3$ holds.

∘ CASE $a\ b$: Let (Q_1, σ_a) be `infer` (Q, Γ, a) and (Q_2, σ_b) be `infer` (Q_1, Γ, b). By induction hypothesis, we have $Q \sqsubseteq Q_1 \sqsubseteq Q_2$ (**1**) and $(Q_1)\ \Gamma \vdash a : \sigma_a$ (**2**) as well as $(Q_2)\ \Gamma \vdash a : \sigma_b$ (**3**). Hence, by WEAKEN on (2) and (1), we have $(Q_2)\ \Gamma \vdash a : \sigma_a$ (**4**).

Let Q_0 be $(Q_2, \alpha_a \geq \sigma_a, \alpha_b \geq \sigma_b, \beta \geq \bot)$. By rules Weaken, Inst and I-Hyp applied once to (3) and once to (4), we get $(Q_0)\ \Gamma \vdash a : \alpha_a$ **(5)** and $(Q_0)\ \Gamma \vdash b : \alpha_b$ **(6)**. Let Q_3 be `unify` $(Q_0, \alpha_a, \alpha_b \to \beta)$. By Lemma 4.4.1, we have $Q_0 \sqsubseteq Q_3$ **(7)** and $(Q_3)\ \alpha_a \equiv \alpha_b \to \beta$ **(8)**. Let (Q_4, Q_5) be $Q_3 \!\uparrow \mathsf{dom}(Q)$. We have $Q_3 \equiv Q_4 Q_5$ **(9)** by Lemma 3.5.2. By Weaken, (7), once on (5) and once on (6), we have $(Q_3)\ \Gamma \vdash a : \alpha_a$ **(10)** and $(Q_3)\ \Gamma \vdash b : \alpha_b$. By Inst applied to (8) and (10), we get $(Q_3)\ \Gamma \vdash a : \alpha_b \to \beta$ **(11)**. By Rule App, we get $(Q_3)\ \Gamma \vdash a\ b : \beta$. By Weaken applied to (9), we get $(Q_4 Q_5)\ \Gamma \vdash a\ b : \beta$. Since $\mathsf{ftv}(\Gamma) \subseteq \mathsf{dom}(Q) \subseteq \mathsf{dom}(Q_4)$, we get by Rule Gen $(Q_4)\ \Gamma \vdash a\ b : \forall\,(Q_5)\ \beta$, which is the expected result. Moreover, we have $Q \sqsubseteq Q_1 \sqsubseteq Q_2 \equiv Q_0 \sqsubseteq Q_3 \equiv Q_4 Q_5 \equiv^{\mathsf{dom}(Q)} Q_4$. Hence, $Q \sqsubseteq Q_4$ holds.

∘ Case `let` $x = a_1$ `in` a_2: Let (Q_1, σ_1) be `infer` (Q, Γ, a_1). By induction hypothesis, we have $Q \sqsubseteq Q_1$ **(1)** and $(Q_1)\ \Gamma \vdash a_1 : \sigma_1$ **(2)**. Let (Q_2, σ_2) be `infer` $(Q_1, \Gamma, x : \sigma_1, a_2)$. By induction hypothesis, we have $Q_1 \sqsubseteq Q_2$ **(3)** and $(Q_2)\ \Gamma, x : \sigma_1 \vdash a_2 : \sigma_2$ **(4)**. By Rule Weaken, (2), and (3), we have $(Q_2)\ \Gamma \vdash a_1 : \sigma_1$ **(5)**. Rule Let applies on (5) and (4) and gives $(Q_2)\ \Gamma \vdash$ `let` $x = a_1$ `in` $a_2 : \sigma_2$. Moreover, by PI-Trans on (1) and (3), we have $Q \sqsubseteq Q_2$. This is the expected result. ■

7.3 Completeness of the algorithm

Lemma 7.3.1 (Completeness of type inference) *The algorithm W^F is complete, that is, if there exists a solution to a type inference problem (Q, Γ, a), then the algorithm* `infer` (Q, Γ, a) *succeeds and returns a principal solution.*

Proof: Let I be $\mathsf{dom}(Q)$. By hypothesis and Lemma 6.2.6, we have a solution (Q_2, σ_2), that is $(Q_2)\ \Gamma \overset{\triangledown}{\vdash} a : \sigma_2$and $Q \sqsubseteq Q_2$ **(1)** hold. Note that $\mathsf{dom}(Q) \subseteq \mathsf{dom}(Q_2)$ **(2)** holds. We have to show that `infer` (Q, Γ, a) succeeds and returns (Q_1, σ_1) such that $(Q_1, \gamma > \sigma_1) \sqsubseteq^{I \cup \{\gamma\}} (Q_2, \gamma > \sigma_2)$ **(3)**. The proof is by induction on the structure of a. We proceed by case analysis on the last rule of the derivation of $(Q_2)\ \Gamma \overset{\triangledown}{\vdash} a : \sigma_2$.

∘ Case Var: Then a is a variable x, σ_2 is $\Gamma(x)$ and `infer` (Q, Γ, a) succeeds and returns $(Q, \Gamma(x))$. Hence, $\sigma_1 = \sigma_2$ and $Q_1 = Q$. Then (3) holds by Property 3.4.2.iii (page 106) and (1).

∘ Case Fun: Then a is $\lambda(x)\ a'$. By hypothesis, $(Q_2 Q'_2)\ \Gamma, x : \tau_0 \overset{\triangledown}{\vdash} a' : \sigma'_2$ **(4)** and $\mathsf{dom}(Q'_2) \cap \mathsf{ftv}(\Gamma) = \emptyset$ hold, and σ_2 is $\forall\,(Q'_2, \beta \geq \sigma'_2)\ \tau_0 \to \beta$ **(5)**. Let α be a fresh type variable outside $\mathsf{dom}(Q_2 Q'_2)$. We have $(Q_2 Q'_2, \alpha = \tau_0)\ \Gamma, x : \alpha \overset{\triangledown}{\vdash} a' : \sigma'_2$ **(6)** by Corollary 6.1.3 (page 153), Strengthen, and (4). We can derive $(Q, \alpha \geq \bot) \sqsubseteq (Q_2 Q'_2, \alpha = \tau_0)$ **(7)** from (1) by Property 3.4.2.iii (page 106) and rules PE-Free, PI-Context-L, and I-Nil. By induction hypothesis on (6) and (7), `infer` $(Q, \alpha > \bot, \Gamma, x : \alpha, a')$ succeeds with (Q'_1, σ'_1) such that $(Q'_1, \gamma \geq \sigma'_1) \sqsubseteq^{I \cup \{\alpha, \gamma\}} (Q_2 Q'_2, \alpha = \tau_0, \gamma \geq \sigma'_2)$ **(8)** holds. Let (Q_3, Q_4) be $Q'_1 \!\uparrow \mathsf{dom}(Q)$. We note that the prefixes $(Q_3, (Q_4, \gamma \geq \sigma'_1))$ and $(Q'_1, \gamma \geq \sigma'_1)\!\uparrow \mathsf{dom}(Q)$ are equal **(9)** by Definition 3.5.1 and since $\gamma \notin \mathsf{dom}(Q)$. Remember that I is a shorthand for $\mathsf{dom}(Q)$. By Lemma 3.6.13 applied to (8), (9), and (2), and taking $\tau = \alpha \to \gamma$, we get

$$(Q_3, \gamma' \geq \forall\,(Q_4, \gamma \geq \sigma_1')\ \alpha \to \gamma) \sqsubseteq^{I \cup \{\gamma'\}} (Q_2, \gamma' \geq \forall\,(Q_2', \alpha = \tau_0, \gamma \geq \sigma_2')\ \alpha \to \gamma) \quad (\underline{\mathbf{10}})$$

The algorithm returns $(Q_3, \forall\,(Q_4, \gamma \geq \sigma_1')\ \alpha \to \gamma)$, that is, Q_1 is Q_3 and σ_1 is $\forall\,(Q_4, \gamma \geq \sigma_1')\ \alpha \to \gamma)$. Additionally, from (5), Eq-Mono* and α-conversion, σ_2 is equivalent to $\forall\,(Q_2', \alpha = \tau_0, \gamma \geq \sigma_2')\ \alpha \to \gamma$. Then (10) is equivalent to

$$(Q_1, \gamma' \geq \sigma_1) \sqsubseteq^{I \cup \{\gamma'\}} (Q_2, \gamma' \geq \sigma_2)$$

This is the expected result.

∘ Case Let: We have a of the form `let` $x = a_1$ `in` a_2. By hypothesis, we have (Q_2) $\Gamma \vdash^{\triangledown} a_1 : \sigma_0$ and $(Q_2)\ \Gamma, x : \sigma_0 \vdash^{\triangledown} a_2 : \sigma_2$ $(\underline{\mathbf{11}})$. By induction hypothesis, `infer` (Q, Γ, a) succeeds and returns (Q_3, σ_3) such that $(Q_3, \gamma \geq \sigma_3) \sqsubseteq^{I \cup \{\gamma\}} (Q_2, \gamma \geq \sigma_0)$. By Lemma 3.6.2 (page 114), there exists a renaming ϕ and a substitution θ both invariant on $I \cup \{\gamma\}$, as well as a prefix Q_0' such that we have $(Q_3, \gamma \geq \sigma_3) \sqsubseteq (\phi(Q_2, \gamma \geq \sigma_0)\underline{\theta}Q_0')$ $(\underline{\mathbf{12}})$. Besides, we have $\mathsf{dom}(Q_0')\ \#\ \mathsf{dom}((Q_3, \gamma \geq \sigma_3)/I \cup \{\gamma\})$, that is, $\mathsf{dom}(Q_0')\ \#\ \gamma \cup \mathsf{dom}(Q_3/I \cup \mathsf{ftv}(\sigma_3))$ $(\underline{\mathbf{13}})$. By Property 3.4.2.i (page 106), (12), and PE-Free, $Q_3 \sqsubseteq (\phi(Q_2, \gamma \geq \sigma_0)\underline{\theta}Q_0')$ $(\underline{\mathbf{14}})$ holds. From (11), we get $(Q_2, \gamma \geq \sigma_0)\ \Gamma, x : \sigma_0 \vdash a_2 : \sigma_2$ by Corollary 6.1.3 (page 153). Hence, by Lemma 6.1.1, we get $(\phi(Q_2, \gamma \geq \sigma_0))\ \phi(\Gamma, x : \sigma_0) \vdash a_2 : \phi(\sigma_2)$. Since ϕ is disjoint from I, ϕ is invariant on Γ, thus this rewrites to $(\phi(Q_2, \gamma \geq \sigma_0))$ $\Gamma, x : \phi(\sigma_0) \vdash a_2 : \phi(\sigma_2)$. By Corollary 6.1.3 (page 153), we have $(\phi(Q_2, \gamma \geq \sigma_0)\underline{\theta}Q_0')$ $\Gamma, x : \phi(\sigma_0) \vdash a_2 : \phi(\sigma_2)$ $(\underline{\mathbf{15}})$. We can derive $(Q_3, \gamma \geq \sigma_3)\ \sigma_3 \sqsubseteq \gamma$ $(\underline{\mathbf{16}})$ by I-Hyp. From (12), (16), and Lemma 3.6.4, we get $(\phi(Q_2), \gamma \geq \phi(\sigma_0), \underline{\theta}Q_0')\ \sigma_3 \sqsubseteq \gamma$. By (13), we have $\gamma \notin \mathsf{dom}(Q_0')$ and $\mathsf{dom}(Q_0')\ \#\ \mathsf{ftv}(\sigma_3)$. Hence, by R-Context-R and Eq-Free, we get $(\phi(Q_2), \gamma \geq \phi(\sigma_0)), \theta(\sigma_3) \sqsubseteq \theta(\gamma)$. Observing that $\theta(\gamma)$ is γ and that $\gamma \notin \mathsf{ftv}(\theta(\sigma_3))$ (by well-formedness of (12)), we get $(\phi(Q_2))\ \theta(\sigma_3) \sqsubseteq \phi(\sigma_0)$ by R-Context-R, Eq-Free, and Eq-Var. By Property 1.7.2.iii (page 59), we get $(\phi(Q_2, \gamma \geq \sigma_0)\underline{\theta}Q_0')\ \theta(\sigma_3) \sqsubseteq \phi(\sigma_0)$. By Eq-Mono, we get $(\phi(Q_2, \gamma \geq \sigma_0)\underline{\theta}Q_0')\ \sigma_3 \sqsubseteq \phi(\sigma_0)$. By Strengthen on (15), we get $(\phi(Q_2, \gamma \geq \sigma_0)\underline{\theta}Q_0')\ \Gamma, x : \sigma_3 \vdash a_2 : \phi(\sigma_2)$ $(\underline{\mathbf{17}})$. By induction hypothesis applied to (17) and (14), `infer` $(Q_3, \Gamma, x : \sigma_3, a_2)$ succeeds with (Q_1', σ_1) such that $(Q_1', \gamma' \geq \sigma_1) \sqsubseteq^{I \cup \{\gamma'\}}$ $(\phi(Q_2, \gamma \geq \sigma_0)\underline{\theta}Q_0', \gamma' \geq \phi(\sigma_2))$ $(\underline{\mathbf{18}})$. We note that $\mathsf{ftv}(\sigma_2) \in \mathsf{dom}(Q_2)$ by well-formedness of (11). Hence, $\mathsf{ftv}(\phi(\sigma_2)) \in \mathsf{dom}(\phi(Q_2))$ $(\underline{\mathbf{19}})$. We have

$$\begin{array}{lll}
 & (\phi(Q_2, \gamma \geq \sigma_0)\underline{\theta}Q_0', \gamma' \geq \phi(\sigma_2)) & \\
\equiv & (\phi(Q_2, \gamma' \geq \sigma_2, \gamma \geq \sigma_0)\underline{\theta}Q_0') & \text{by (19) and Eq-Comm} \\
\equiv^{I \cup \{\gamma'\}} & (\phi(Q_2, \gamma' \geq \sigma_2)) & \text{by Eq-Free} \\
\equiv^{I \cup \{\gamma'\}} & (\phi(Q_2, \gamma' \geq \sigma_2))\underline{\phi} & \text{by Eq-Free} \\
\equiv^{I \cup \{\gamma'\}} & (Q_2, \gamma' \geq \sigma_2) & \text{by Property 3.4.2.iv (page 106)}
\end{array}$$

Hence, $(\phi(Q_2, \gamma \geq \sigma_0)\underline{\theta}Q_0', \gamma' \geq \phi(\sigma_2)) \equiv^{I \cup \{\gamma'\}} (Q_2, \gamma' \geq \sigma_2)$ holds, and we get the expected result from (18), that is,

$$(Q_1', \gamma' \geq \sigma_1) \sqsubseteq^{I \cup \{\gamma'\}} (Q_2, \gamma' \geq \sigma_2)$$

- Case App: By hypothesis, we have

$$(Q_2)\ \Gamma \overset{\triangledown}{\vdash} a_1 : \sigma_a\ (\underline{\mathbf{20}}) \qquad (Q_2)\ \Gamma \overset{\triangledown}{\vdash} a_2 : \sigma_b\ (\underline{\mathbf{21}}) \qquad (Q_2)\ \sigma_a \sqsubseteq \forall\,(Q_3)\ \tau_2 \to \tau_1\ \ (\underline{\mathbf{22}})$$

$$(Q_2)\ \sigma_b \sqsubseteq \forall\,(Q_3)\ \tau_2\ \ (\underline{\mathbf{23}}) \qquad\qquad \sigma_2 = \forall\,(Q_3)\ \tau_1$$

Let $\alpha_a, \alpha_b, \beta$ be fresh variables. Let I_a be $I \cup \{\alpha_a\}$, and I_b be $I \cup \{\alpha_b\}$. By induction hypothesis, (20), and (1), $\mathtt{infer}\ (Q_{,}\Gamma, a_1)$ succeeds and returns (Q_a, σ'_a) such that $(Q_a, \alpha_a \geq \sigma'_a) \sqsubseteq^{I_a} (Q_2, \alpha_a \geq \sigma_a)$ holds. By Lemma 3.6.2 (page 114), there exists a renaming ϕ_a and a substitution θ_a, both invariant on I_a, as well as Q'_0 such that we have $(Q_a, \alpha_a \geq \sigma'_a) \sqsubseteq (\phi_a(Q_2, \alpha_a \geq \sigma_a)\underline{\theta_a}Q'_0)$ $(\underline{\mathbf{24}})$ holds. Moreover, we have $\mathsf{dom}(Q'_0)\ \#\ \mathsf{dom}(Q_a/I \cup \mathsf{ftv}(\sigma'_a))$ $(\underline{\mathbf{25}})$. We have $Q_a \sqsubseteq (\phi_a(Q_2, \alpha_a \geq \sigma_a)\underline{\theta_a}Q'_0)$ $(\underline{\mathbf{26}})$ from (24) and PE-Free. Let I' be $\mathsf{dom}(Q_a)$ $(\underline{\mathbf{27}})$. By Lemmas 6.1.1 and 6.1.3, and by (21), we get $(\phi_a(Q_2, \alpha_a \geq \sigma_a)\underline{\theta_a}Q'_0)\ \Gamma \vdash a_2 : \phi_a(\sigma_b)$ $(\underline{\mathbf{28}})$. Hence, by induction hypothesis, (26) and (28), $\mathtt{infer}\ (Q_a, \Gamma, a_2)$ succeeds and returns (Q_b, σ'_b) such that $(Q_b, \alpha_b \geq \sigma'_b) \sqsubseteq^{I' \cup \{\alpha_b\}} (\phi_a(Q_2, \alpha_a \geq \sigma_a)\underline{\theta_a}Q'_0, \alpha_b \geq \phi_a(\sigma_b))$ $(\underline{\mathbf{29}})$ holds. Let ϕ' be a renaming of α_a to a fresh variable α'_a (that is, not in the domain or in the free variables of Q_1, Q_2, ϕ_a, θ_a, Q'_0). Let θ'_a be the substitution extracted from $\phi'(\underline{\theta_a})$ and Q''_0 be $\phi'(Q''_0)$. We remark from (21) that $\mathsf{ftv}(\sigma_b) \subseteq \mathsf{dom}(Q_2)$. Hence, $\mathsf{ftv}(\phi_a(\sigma_b)) \subseteq \mathsf{dom}(\phi_a(Q_2))$ $(\underline{\mathbf{30}})$, thus $\alpha_a \notin \mathsf{ftv}(\phi_a(\sigma_b))$. As a consequence, $\phi \circ \phi_a(\sigma_b) = \phi_a(\sigma_b)$ $(\underline{\mathbf{31}})$. By Property 3.4.2.iv (page 106) (using the renaming ϕ), and (31), we get $(\phi_a(Q_2, \alpha_a \geq \sigma_a)\underline{\theta_a}Q'_0, \alpha_b \geq \phi_a(\sigma_b)) \equiv^{I' \cup \{\alpha_b\}} (\phi_a(Q_2, \alpha'_a \geq \sigma_a)\underline{\theta'_a}Q''_0, \alpha_b \geq \phi_a(\sigma_b))\underline{\phi}$ $(\underline{\mathbf{32}})$. Additionally, α_a was chosen such that $\alpha_a \notin I' \cup \{\alpha_b\}$ $(\underline{\mathbf{33}})$. Hence, by PE-Free, (32), and (33), we get

$$(\phi_a(Q_2, \alpha_a \geq \sigma_a)\underline{\theta_a}Q'_0, \alpha_b \geq \phi_a(\sigma_b)) \equiv^{I' \cup \{\alpha_b\}} (\phi_a(Q_2, \alpha'_a \geq \sigma_a)\underline{\theta'_a}Q''_0, \alpha_b \geq \phi_a(\sigma_b))$$

With (29) and PI-Trans, we get

$$(Q_b, \alpha_b \geq \sigma'_b) \sqsubseteq^{I' \cup \{\alpha_b\}} (\phi_a(Q_2, \alpha'_a \geq \sigma_a)\underline{\theta'_a}Q''_0, \alpha_b \geq \phi_a(\sigma_b))$$

Observing that we have $\mathsf{utv}(\sigma'_a) \subseteq I'$ from (24) and (27), we get by Property 3.4.2.iii (page 106)

$$(Q_b, \alpha_b{\geq}\sigma'_b, \alpha_a{\geq}\sigma'_a, \beta{\geq}\bot) \sqsubseteq^{I' \cup \{\alpha_b, \alpha_a, \beta\}} (\phi_a(Q_2, \alpha'_a{\geq}\sigma_a)\underline{\theta'_a}Q''_0, \alpha_b{\geq}\phi_a(\sigma_b), \alpha_a{\geq}\sigma'_a, \beta{\geq}\bot)$$

By Property 3.4.2.i (page 106), we get

$$(Q_b, \alpha_b{\geq}\sigma'_b, \alpha_a{\geq}\sigma'_a, \beta{\geq}\bot) \sqsubseteq^{I \cup \{\alpha_b, \alpha_a\}} (\phi_a(Q_2, \alpha'_a{\geq}\sigma_a)\underline{\theta'_a}Q''_0, \alpha_b{\geq}\phi_a(\sigma_b), \alpha_a{\geq}\sigma'_a, \beta{\geq}\bot)$$

Let Q_4 be $(Q_b, \alpha_a \geq \sigma'_a, \alpha_b \geq \sigma'_b, \beta \geq \bot)$, J be $I \cup \{\alpha_b, \alpha_a, \beta\}$, and P_4 be $(\phi_a(Q_2, \alpha'_a \geq \sigma_a)\underline{\theta'_a}Q''_0, \alpha_b \geq \phi_a(\sigma_b), \alpha_a \geq \sigma'_a, \beta \geq \bot)$. The above relation can be written

$$Q_4 \sqsubseteq^J P_4\ (\underline{\mathbf{34}})$$

We have $(Q_a, \alpha_a \geq \sigma'_a)\ \sigma'_a \sqsubseteq \alpha_a$, thus by Lemma 3.6.4 applied to (24), we can derive $(\phi_a(Q_2, \alpha_a \geq \sigma_a)\underline{\theta_a}Q'_0)\ \sigma'_a \sqsubseteq \alpha_a$. Observing that $\alpha_a \notin \mathsf{ftv}(\sigma'_a)$, we get $(\phi_a(Q_2, \alpha'_a \geq \sigma_a)\underline{\theta'_a}Q''_0)\ \sigma'_a \sqsubseteq \alpha'_a$ by Property 1.7.2.i (page 59). Hence, we have

$$P_4 \sqsubseteq (\phi_a(Q_2, \alpha'_a \geq \sigma_a)\underline{\theta'_a}Q''_0, \alpha_b \geq \phi_a(\sigma_b), \alpha_a = \alpha'_a, \beta \geq \bot)\ \ (\underline{\mathbf{35}})$$

and we write P_5 for the right-hand side. By PE-Swap and PE-Commut, using (30), we get

$$P_5 \equiv (\phi_a(Q_2), \alpha_a \geq \phi_a(\sigma_a), \alpha_b \geq \phi_a(\sigma_b), \beta \geq \bot, \alpha'_a = \alpha_a, \underline{\theta'_a} Q''_0) \quad (\underline{\mathbf{36}})$$

and we write P_6 for the right-hand side. By Property 1.7.2.i (page 59) applied to (22) and (23), we get $(\phi_a(Q_2))\ \phi_a(\sigma_a) \sqsubseteq \phi_a(\forall\,(Q_3)\ \tau_2 \to \tau_1)$ and $(\phi_a(Q_2))\ \phi_a(\sigma_b) \sqsubseteq \phi_a(\forall\,(Q_3)\ \tau_2)$. Hence, we can derive

$$P_6 \sqsubseteq (\phi_a(Q_2), \alpha_a \geq \phi_a(\forall\,(Q_3)\ \tau_2 \to \tau_1), \alpha_b \geq \phi_a(\forall\,(Q_3)\ \tau_2), \beta \geq \bot, \alpha'_a = \alpha_a, \underline{\theta'_a} Q''_0) \quad (\underline{\mathbf{37}})$$

and we write P_7 for the right-hand side. By PE-Free and I-Drop* (twice on $\phi_a(Q_3)$), we can derive

$$P_7 \sqsubseteq (\phi_a(Q_2 Q_3), \alpha_a = \phi_a(\tau_2 \to \tau_1), \alpha_b = \phi_a(\tau_2), \beta \geq \bot, \alpha'_a = \alpha_a, \underline{\theta'_a}) \quad (\underline{\mathbf{38}})$$

and we write P_8 for the right-hand side. Finally, by Property 3.4.2.iv (page 106) and I-Bot, we get

$$P_8 \sqsubseteq^J (Q_2 Q_3, \alpha_a = \tau_2 \to \tau_1, \alpha_b = \tau_2, \beta = \tau_1) \quad (\underline{\mathbf{39}})$$

Let Q_6 be $(Q_2 Q_3, \alpha_a = \tau_2 \to \tau_1, \alpha_b = \tau_2, \beta = \tau_1)$. By (34), (35), (36), (37), (38), and (39), we get $Q_4 \sqsubseteq^J Q_6$. Moreover, we have $(Q_6)\ \alpha_a \equiv \alpha_b \to \beta$ by lemma 1.5.10.

By completeness of unification, (Lemma 4.6.2), `unify` $(Q_4)\ \alpha_a \doteq \alpha_b \to \beta$ succeeds and returns Q_5 such that we have $Q_5 \sqsubseteq^J Q_6$. Let (Q_7, Q_8) be $Q_5 \uparrow \mathsf{dom}(Q)$. By hypothesis, $I = \mathsf{dom}(Q)$, and $I \subseteq \mathsf{dom}(Q_2)$. By Lemma 3.6.13, we have a derivation of

$$(Q_7, \gamma \geq \forall\,(Q_8)\ \beta) \sqsubseteq^{I \cup \{\gamma\}} (Q_2, \gamma \geq \forall\,(Q_3)\ \tau_1)$$

This is the expected result.

○ Case Oracle cannot appear since type inference is only performed in ML^{F}, not in $\mathrm{ML}^{\mathrm{F}}_{\star}$. ■

7.4 Decidability of type inference

Lemma 7.4.1 *The type inference algorithm always terminates, either by failing or by returning a prefix and a type scheme.*

Proof: By induction on the size of the expression being typed. All cases are straightforward, using Lemma 4.5.5 for the termination of `unify`. ■

Thanks to Lemmas 7.2.1 and 7.3.1, the type inference algorithm is proved sound and complete. Additionally, the algorithm always terminates by Lemma 7.4.1. Hence, the following theorem:

Theorem 4 (Type inference) *The set of solutions of a solvable type inference problem admits a principal solution. Given any type inference problem, the algorithm* W^F *either returns a principal solution or fails if no solution exists.*

Chapter 8

Type annotations

In this chapter, we restrict our attention to ML^F, *i.e.* to expressions that do not contain oracles. Since expressions of ML^F are exactly those of ML, the only difference lies in the richer types and typing rules. However, this is not sufficient to provide usable first-class polymorphism, since ML^F without type annotations and ML are equivalent, as shown by the next section. The following sections introduce type annotations as primitives and show how to use them to introduce polymorphism in ML^F.

8.1 ML^F without type annotations

In the following, we write $\sqsubseteq_{ML}$ for the ML generalized instance relation. It allows instantiation of quantified variables with any monotype and immediate generalization of freshly introduced variables. We define it as follows:

$$\forall\,(\bar\alpha)\ \sigma \sqsubseteq_{ML} \forall\,(\bar\beta)\ \sigma[\bar\tau/\bar\alpha] \text{ holds if and only if } \bar\beta \text{ is disjoint from } \mathsf{ftv}(\sigma).$$

The following property is standard in ML:

Lemma 8.1.1 *If we have* $\forall\,(\bar\alpha)\ \tau_1 \sqsubseteq_{ML} \forall\,(\bar\beta)\ \tau_2$*, then for any* τ *such that* $\mathsf{ftv}(\tau) \# \bar\alpha$ *and* $\mathsf{ftv}(\tau) \# \bar\beta$*, we have* $\forall\,(\bar\alpha)\ \tau[\tau_1/\gamma] \sqsubseteq_{ML} \forall\,(\bar\beta)\ \tau[\tau_2/\gamma]$

Proof: By definition, we have $\bar\beta \# \mathsf{ftv}(\tau_1)$, and $\tau_2 = \tau_1[\bar\tau/\bar\alpha]$ **(1)**. By hypothesis, we have $\bar\beta \# \mathsf{ftv}(\tau)$, thus $\bar\beta \# \mathsf{ftv}(\tau[\tau_1/\gamma])$. As a consequence, we have $\forall\,(\bar\alpha)\ \tau[\tau_1/\gamma] \sqsubseteq_{ML} \forall\,(\bar\beta)\ \tau[\tau_1/\gamma][\bar\tau/\bar\alpha]$. By hypothesis, we have $\bar\alpha \# \mathsf{ftv}(\tau)$ **(2)**. We get the expected result by observing that $\forall\,(\bar\beta)\ \tau[\tau_1/\gamma][\bar\tau/\bar\alpha]$ is equal to $\forall\,(\bar\beta)\ \tau[\tau_2/\gamma]$ by (1) and (2). ■

Polytypes that do not contain rigid bindings are said to be *flexible*. Let `flex` be the function defined on polytypes and prefixes that transforms every rigid binding into a flexible binding. For instance, $\texttt{flex}\ (\forall\,(\alpha = \sigma_1, \beta \geq \sigma_2)\ \sigma)$ is $\forall\,(\alpha \geq \texttt{flex}\ (\sigma_1), \beta \geq \texttt{flex}\ (\sigma_2))\ \texttt{flex}\ (\sigma)$. We say that a derivation is *flexible* if it does not contain any rigid bindings in types or in the prefix. A judgment is flexible if it has a flexible derivation.

Properties 8.1.2

i) *If* $(Q)\ \sigma_1 \equiv \sigma_2$*, then* $(\texttt{flex}\ (Q))\ \texttt{flex}\ (\sigma_1) \equiv \texttt{flex}\ (\sigma_2)$ *is flexible.*

ii) *If* $(Q)\ \sigma_1 \mathrel{-\!\!\!\!\sqsubseteq} \sigma_2$*, then* $(\texttt{flex}\ (Q))\ \texttt{flex}\ (\sigma_1) \sqsubseteq \texttt{flex}\ (\sigma_2)$ *is flexible.*

iii) *If* $(Q)\ \sigma_1 \sqsubseteq \sigma_2$*, then* $(\texttt{flex}\ (Q))\ \texttt{flex}\ (\sigma_1) \sqsubseteq \texttt{flex}\ (\sigma_2)$ *is flexible.*

Proof: For each property, by induction on the derivation. Equivalence and Instance cases are easy. As for the abstraction relation, the cases A-Hyp and R-Context-Rigid are replaced by (respectively) I-Hyp and R-Context-Flexible. Finally, I-Rigid is replaced by Eq-Refl and I-Equiv*. ■

We lift the function $\texttt{flex}$ to typing environments and typing judgments in the natural way. This operation is correct in the following sense:

Lemma 8.1.3 *If* $(Q)\ \Gamma \vdash a : \sigma$ *holds in ML*^{F}*, then so does* $\texttt{flex}\ ((Q)\ \Gamma \vdash a : \sigma)$*.*

Proof: By induction on the derivation of $(Q)\ \Gamma \vdash a : \sigma$. Case Var is immediate. Cases App, Fun and Let are by induction hypothesis. Case Inst is by Property 8.1.2.iii. Case Gen: We have $(Q)\ \Gamma \vdash a : \forall\,(\alpha \diamond \sigma_1)\ \sigma_2$, and the premise is $(Q, \alpha \diamond \sigma_1) \vdash a : \sigma_2$, with $\alpha \notin \mathsf{ftv}(\Gamma)$. Note that $\alpha \notin \mathsf{ftv}(\texttt{flex}\ (\Gamma))$. By induction hypothesis, $(\texttt{flex}\ (Q), \alpha \geq \texttt{flex}\ (\sigma_1))\ \texttt{flex}\ (\Gamma) \vdash a : \texttt{flex}\ (\sigma_2)$ holds. Hence, $(\texttt{flex}\ (Q))\ \texttt{flex}\ (\Gamma) \vdash a : \forall\,(\alpha \geq \texttt{flex}\ (\sigma_1))\ \texttt{flex}\ (\sigma_2)$ holds by Gen. By definition, this means $(\texttt{flex}\ (Q))\ \texttt{flex}\ (\Gamma) \vdash a : \texttt{flex}\ \forall\,(\alpha \diamond \sigma_1)\ \sigma_2$. This is the expected result. ■

Definition 8.1.4 We define the ML approximation $\langle\!\langle \sigma \rangle\!\rangle$ of an ML^F flexible type σ as follows:

$$\langle\!\langle \bot \rangle\!\rangle = \forall\,(\alpha)\ \alpha \qquad\qquad \langle\!\langle \tau \rangle\!\rangle = \tau \qquad\qquad \frac{\langle\!\langle \sigma \rangle\!\rangle = \forall\,(Q)\ \tau}{\langle\!\langle \forall\,(\alpha \geq \sigma)\ \sigma' \rangle\!\rangle = \forall\,(Q)\ \langle\!\langle \sigma' \rangle\!\rangle[\tau/\alpha]}$$

We extend $\langle\!\langle \cdot \rangle\!\rangle$ to prefixes by returning a pair of an unconstrained prefix and a substitution:

$$\langle\!\langle \emptyset \rangle\!\rangle = \emptyset, \texttt{id} \qquad\qquad \frac{\langle\!\langle \sigma \rangle\!\rangle = \forall\,(Q_1)\ \tau_1 \qquad \langle\!\langle Q' \rangle\!\rangle = Q_2, \theta_2}{\langle\!\langle (\alpha \diamond \sigma, Q') \rangle\!\rangle = Q_1 Q_2, [\tau_1/\alpha] \circ \theta_2}$$

□

Note that ML types are invariant by $\langle\!\langle\cdot\rangle\!\rangle$. Indeed, we have $\langle\!\langle\forall\,(\bar\alpha)\ \tau\rangle\!\rangle = \forall\,(\bar\alpha)\ \tau$. Note also that if $\langle\!\langle\sigma\rangle\!\rangle$ is $\forall\,(Q)\ \tau$, then Q is unconstrained.

Properties 8.1.5

- ***i)*** *For any σ and any substitution θ, we have $\langle\!\langle\theta(\sigma)\rangle\!\rangle = \theta(\langle\!\langle\sigma\rangle\!\rangle)$.*
- ***ii)*** *The following implications hold:*

$$\frac{\langle\!\langle Q\rangle\!\rangle = Q', \theta}{\langle\!\langle\forall\,(Q)\ \sigma\rangle\!\rangle = \forall\,(Q')\ \theta(\langle\!\langle\sigma\rangle\!\rangle)} \qquad \frac{\langle\!\langle Q_1\rangle\!\rangle = Q'_1, \theta_1 \qquad \langle\!\langle Q_2\rangle\!\rangle = Q'_2, \theta_2}{\langle\!\langle Q_1 Q_2\rangle\!\rangle = Q'_1 Q'_2, \theta_1 \circ \theta_2}$$

- ***iii)*** *We have $\langle\!\langle\sigma\rangle\!\rangle/ = \sigma/$ for any σ.*
- ***iv)*** *If $\sigma \equiv \tau$ holds, then $\langle\!\langle\sigma\rangle\!\rangle = \forall\,(Q)\ \tau$ with $\mathsf{dom}(Q) \# \mathsf{ftv}(\tau)$.*
- ***v)*** *If $\langle\!\langle Q\rangle\!\rangle$ is (Q', θ), then there exists θ' such that θ is $\theta' \circ \widehat{Q}$.*
- ***vi)*** *If $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ is flexible, and $\langle\!\langle Q\rangle\!\rangle = (Q', \theta)$, then $\theta(\langle\!\langle\sigma_1\rangle\!\rangle) \sqsubseteq_{ML} \theta(\langle\!\langle\sigma_2\rangle\!\rangle)$ holds.*

See proof in the Appendix (page 293).

We lift $\langle\!\langle\cdot\rangle\!\rangle$ to typing environments in the obvious way.

Lemma 8.1.6 *If we have a flexible derivation of $(Q)\ \Gamma \vdash a : \sigma$, then we have a derivation of $\theta(\langle\!\langle\Gamma\rangle\!\rangle) \vdash a : \theta(\langle\!\langle\sigma\rangle\!\rangle)$ in ML, where $\langle\!\langle Q\rangle\!\rangle$ is Q', θ.*

Proof: By induction on the derivation. Case VAR is immediate. Cases FUN, APP, and LET are by induction hypothesis. Case INST is a direct consequence of Property 8.1.5.vi. Case GEN: The premise is $(Q, \alpha \diamond \sigma_1)\ \Gamma \vdash a : \sigma_2$. Let $\forall\,(Q_1)\ \tau_1$ be $\langle\!\langle\sigma_1\rangle\!\rangle$, and θ_1 be $[\tau_1/\alpha]$. We chose Q_1 such that $\mathsf{dom}(Q_1) \# \mathsf{ftv}(\Gamma)$ (**1**). By Property 8.1.5.ii, we have $\langle\!\langle(Q, \alpha \diamond \sigma_1)\rangle\!\rangle = Q'Q_1, \theta \circ \theta_1$. By induction hypothesis, we have $\theta \circ \theta_1(\langle\!\langle\Gamma\rangle\!\rangle) \vdash_{ML} a : \theta \circ \theta_1(\langle\!\langle\sigma_2\rangle\!\rangle)$. Since $\alpha \notin \mathsf{ftv}(\Gamma)$, we have $\theta_1(\langle\!\langle\Gamma\rangle\!\rangle) = \langle\!\langle\Gamma\rangle\!\rangle$. Hence, $\theta(\langle\!\langle\Gamma\rangle\!\rangle) \vdash_{ML} a : \theta \circ \theta_1(\langle\!\langle\sigma_2\rangle\!\rangle$ holds. From (1), we get by Rule GEN of ML, $\theta(\langle\!\langle\Gamma\rangle\!\rangle) \vdash_{ML} a : \forall\,(Q_1)\ \theta \circ \theta_1(\langle\!\langle\sigma_2\rangle\!\rangle)$. This is $\theta(\langle\!\langle\Gamma\rangle\!\rangle) \vdash_{ML} a : \theta(\forall\,(Q_1)\ \theta_1(\langle\!\langle\sigma_2\rangle\!\rangle))$. This is the expected result, by observing that $\langle\!\langle\forall\,(\alpha \diamond \sigma_1)\ \sigma_2\rangle\!\rangle$ is $\forall\,(Q_1)\ \theta_1(\langle\!\langle\sigma_2\rangle\!\rangle)$. ■

Corollary 8.1.7 *If an expression a is typable under a flexible typing environment Γ in ML^F, then a is typable under $\langle\!\langle\Gamma\rangle\!\rangle$ in ML.*

Proof: Direct consequence of Lemmas 8.1.6 and 8.1.3. ■

The inverse inclusion has already been stated in Section 5.2.1. In the particular case where the initial typing context Γ_0 contains only ML types, a closed expression can be typed in ML^F under Γ_0 if and only if it can be typed in ML under Γ_0. This is not true for ML$^F_\star$ in which the expression $\lambda(x)\ (x : \star)\ x$ is typable. Indeed, as shown in Chapter 9, all terms of System F can be typed in ML$^F_\star$.

8.2 Introduction to type annotations

The following example, which describes a single annotation, should provide intuition for the general case.

Example 8.2.14 Let f be a constant of type $\sigma = \forall\,(\alpha{=}\sigma_{\mathtt{id}}, \alpha'{\geq}\sigma_{\mathtt{id}})\ \alpha \to \alpha'$ with the δ-reduction $f\ v \longrightarrow (v : \star)$. Then the expression a defined as $\lambda(x)\ (f\ x)\ x$ behaves like $\lambda(x)\ x\ x$ and is well-typed, of type $\forall\,(\alpha{=}\sigma_{\mathtt{id}})\ \alpha \to \alpha$. To see this, let Q and Γ stand for $(\alpha{=}\sigma_{\mathtt{id}}, \alpha'{\geq}\sigma_{\mathtt{id}})$ and $x : \alpha$. By rules Inst, Var, and App $(Q)\ \Gamma \vdash f\ x : \alpha'$ holds; hence by Rule Gen, we get $(\alpha = \sigma_{\mathtt{id}})\ \Gamma \vdash f\ x : \forall\,(\alpha' \geq \sigma_{\mathtt{id}})\ \alpha'$ since α' is not free in Γ. By Rule Eq-Var, we have $\forall\,(\alpha' \geq \sigma_{\mathtt{id}})\ \alpha' \equiv \sigma_{\mathtt{id}}$ (under any prefix); besides, $\sigma_{\mathtt{id}} \sqsubseteq \alpha \to \alpha$ holds under any prefix that binds α. Thus, we get $(\alpha = \sigma_{\mathtt{id}})\ \Gamma \vdash f\ x : \alpha \to \alpha$ by Rule Inst. The result follows by rules App, Fun, and Gen.

Observe that the static effect of f in $f\ x$ is (i) to force the type of x to be abstracted by a type variable α bound to $\sigma_{\mathtt{id}}$ in Q and (ii) to allow $f\ x$, that is x, to have the type $\sigma_{\mathtt{id}}$, exactly as the oracle $(x : \star)$ would. Notice that the bound of α in σ is rigid: the function f expects a value v that must have type $\sigma_{\mathtt{id}}$, and not an instance of $\sigma_{\mathtt{id}}$. Conversely, the bound of α' is flexible: the type of $f\ v$ is $\sigma_{\mathtt{id}}$ but may also be any instance of $\sigma_{\mathtt{id}}$.

Note that a behaves as the auto-application function $\lambda(x)\ x\ x$ (when applied to the same value, reduction steps of the latter can be put in correspondence with major reduction steps of the former), which is not typable (see Example 6.2.12).

8.3 Annotation primitives

Definition 8.3.1 We call *annotations* the denumerable collection of unary primitives $(\exists\,(Q)\ \sigma)$, defined for all prefixes Q and polytypes σ closed under Q. The initial typing environment Γ_0 contains these primitives with their associated type:

$$(\exists\,(Q)\ \sigma) : \forall\,(Q)\ \forall\,(\alpha = \sigma)\ \forall\,(\beta \geq \sigma)\ \alpha \to \beta \qquad \in \Gamma_0$$

We may identify annotation primitives up to the equivalence of their types. □

Besides, we write $(a : \exists\,(Q)\ \sigma)$ for the application $(\exists\,(Q)\ \sigma)\ a$. We also abbreviate $(\exists\,(Q)\ \sigma)$ as (σ) when all bounds in Q are unconstrained. Actually, replacing an annotation $(\exists\,(Q)\ \sigma)$ by (σ) preserves typability and, more precisely, preserves typings.

Note the difference between $(\exists\,(Q)\ \sigma)$ and $(\forall\,(Q)\ \sigma)$. The latter would require the argument to have type $\forall\,(Q)\ \sigma$ while the former only requires the argument to have type σ under some prefix Q' to be determined from the context.

Example 8.3.15 As seen in Example 6.2.12, $\lambda(x)\ x\ x$ is not typable in ML$^{\text{F}}$. We also claimed that it would be typable with annotations. We give a typing derivation for $\lambda(x)\ \texttt{let}\ y = (x : \sigma_{\texttt{id}})\ \texttt{in}\ y\ y$. We write a for this expression and σ_a for $\forall\,(\alpha = \sigma_{\texttt{id}})\ \forall\,(\alpha' \geq \sigma_{\texttt{id}})\ \alpha \to \alpha'$.

$$\textsc{Gen}\ \dfrac{\textsc{Fun}\ \dfrac{\textsc{Let}\ \dfrac{\begin{array}{c}(\alpha = \sigma_{\texttt{id}})\ \Gamma, x : \alpha \vdash (x : \sigma_{\texttt{id}}) : \sigma_{\texttt{id}}\quad (\underline{\mathbf{2}})\\ (\alpha = \sigma_{\texttt{id}})\ \Gamma, x : \alpha, y : \sigma_{\texttt{id}} \vdash y\ y : \sigma_{\texttt{id}}\quad (\underline{\mathbf{1}})\end{array}}{(\alpha = \sigma_{\texttt{id}})\ \Gamma, x : \alpha \vdash \texttt{let}\ y = (x : \sigma_{\texttt{id}})\ \texttt{in}\ y\ y : \sigma_{\texttt{id}}}}{(\alpha = \sigma_{\texttt{id}})\ \Gamma \vdash a : \forall\,(\alpha' \geq \sigma_{\texttt{id}})\ \alpha \to \alpha'}}{(\emptyset)\ \Gamma \vdash a : \sigma_a}$$

The judgment (1) can be easily derived, since y is known to have the polymorphic type $\sigma_{\texttt{id}}$. The key point of this derivation is how we get the judgment (2). That is, having $x : \alpha$ in the typing environment, and $(\alpha = \sigma_{\texttt{id}})$ in the prefix, how can we derive $(Q)\ \Gamma \vdash (x : \sigma_{\texttt{id}}) : \sigma_{\texttt{id}}$? Here is the derivation, writing Q for $(\alpha = \sigma_{\texttt{id}})$:

$$\textsc{App}\ \dfrac{\textsc{Inst}\ \dfrac{(Q)\ \Gamma \vdash (\sigma_{\texttt{id}}) : \forall\,(\alpha_1 = \sigma_{\texttt{id}})\ \forall\,(\alpha_2 = \sigma_{\texttt{id}})\ \alpha_1 \to \alpha_2}{(Q)\ \Gamma \vdash (\sigma_{\texttt{id}}) : \forall\,(\alpha_2 = \sigma_{\texttt{id}})\ \alpha \to \alpha_2} \qquad (Q)\ \Gamma \vdash x : \alpha}{(Q)\ \Gamma \vdash (x : \sigma_{\texttt{id}}) : \sigma_{\texttt{id}}}$$

The instantiation simply shares α_1 with α in the prefix (rules A-Hyp and R-Context-Rigid).

While annotations have been introduced as primitives for simplicity of presentation, they are obviously meant to be applied. Notice that the type of an annotation may be instantiated before the annotation is applied. However, the annotation keeps exactly the same “revealing power” after instantiation. This is described by the following technical lemma (the reader may take $\emptyset$ for Q_0 at first).

Lemma 8.3.2 *We have* $(Q_0)\ \Gamma \vdash (a : \exists\,(Q)\ \sigma) : \sigma_0$ *iff there exists a type* $\forall\,(Q')\ \sigma'_1$ *such that* $(Q_0)\ \Gamma \vdash a : \forall\,(Q')\ \sigma'_1$ *holds together with the following relations:*

$$Q_0 Q \sqsubseteq Q_0 Q' \qquad\qquad (Q_0 Q')\ \sigma'_1 \mathrel{\exists} \sigma \qquad\qquad (Q_0)\ \forall\,(Q')\ \sigma \sqsubseteq \sigma_0$$

The prefix Q of the annotation $\exists\,(Q)\ \sigma$ may be instantiated into Q'. However, Q' guards $\sigma'_1 \mathrel{\exists} \sigma$ in $(Q_0 Q')\ \sigma'_1 \mathrel{\exists} \sigma$. In particular, the lemma would not hold with $(Q_0)\ \forall\,(Q')\ \sigma'_1 \mathrel{\exists} \forall\,(Q'')\ \sigma$ and $(Q_0)\ \forall\,(Q'')\ \sigma'_1 \sqsubseteq \sigma_0$. Lemma 8.3.2 has similarities with Rule Annot of Poly-ML [GR99].
See proof in the Appendix (page 295).

Corollary 8.3.3 *The judgment* $(Q)\ \Gamma \vdash (a : \star) : \sigma_0$ *holds iff there exists an annotation* (σ) *such that* $(Q)\ \Gamma \vdash (a : \sigma) : \sigma_0$ *holds.*

See proof in the Appendix (page 296).

Hence, all expressions typable in $\mathrm{ML}^{\mathrm{F}}_{\star}$ are typable in ML^{F} as long as all annotation primitives are in the initial typing context Γ_0. Conversely, the annotation $(\exists\,(Q)\ \sigma)$ can be simulated by $\lambda(x)\ (x : \star)$ in $\mathrm{ML}^{\mathrm{F}}_{\star}$, both statically and dynamically. Hence annotation primitives are unnecessary in $\mathrm{ML}^{\mathrm{F}}_{\star}$.

Reduction of annotations The δ-reduction for annotations just replaces explicit type information by oracles.

$$(v : \exists\,(Q)\ \sigma) \longrightarrow (v : \star)$$

Lemma 8.3.4 (Soundness of type annotations) *Type annotation primitives validate the required hypotheses* **H0** *(arity),* **H1** *(subject-reduction), and* **H2** *(progress).*

See proof in the Appendix (page 296).

Syntactic sugar As mentioned earlier, $\lambda(x : \sigma)\ a$ is syntactic sugar for $\lambda(x)$ `let` $x = (x : \sigma)$ `in` a. The derived typing rule is:

$$\textsc{Fun}^{\star}\qquad \frac{(Q)\ \Gamma, x : \sigma \vdash a : \sigma' \qquad Q' \sqsubseteq Q}{(Q)\ \Gamma \vdash \lambda(x : \exists\,(Q')\ \sigma)\ a : \forall\,(\alpha = \sigma)\ \forall\,(\alpha' \geq \sigma')\ \alpha \to \alpha'}$$

This rule is actually simpler than the derived annotation rule suggested by Lemma 8.3.2, because instantiation is here left to each occurrence of the annotated program variable x in a.

The derived reduction rule is $(\lambda(x : \exists\,(Q)\ \sigma)\ a)\ v \xrightarrow{\beta_\star}$ `let` $x = (v : \exists\,(Q)\ \sigma)$ `in` a. Indeed, values must then be extended with expressions of the form $\lambda(x : \exists\,(Q)\ \sigma)\ a$.

Part III

Expressiveness of ML^{F}

We have already seen that all ML programs can be written in ML^{F} without annotations. In Chapter 9, we provide a straightforward compositional translation of terms of System F into ML^{F}. In Chapter 10, we then identify a subsystem of ML^{F}, called Shallow ML^{F}, whose let-binding-free version is exactly the target of the encoding of System F. We give an interpretation of types of Shallow ML^{F} that allows every typing in Shallow ML^{F} to be transformed into one or many typings in System F. In Chapter 11 we consider some extensions of ML^{F}, including references and propagation of type annotations. Finally, Chapter 12 takes a closer look at the ML^{F} programming language by giving some examples of programs that make use of first-class polymorphism.

Chapter 9

Encoding System F into $\mathrm{ML^F}$

In this chapter, we prove that System F can be mechanically encoded into $\mathrm{ML^F}$ with type annotations primitives (which we simply call $\mathrm{ML^F}$). We show that the translation is correct, that is, if a term typable is in System F, then its translation is typable in $\mathrm{ML^F}$. This result illustrates the expressiveness of $\mathrm{ML^F}$: it shows that all System F expressions are indeed available in $\mathrm{ML^F}$. Moreover, thanks to the results of Chapter 7, we know that the translated terms contain all the needed type information to make type inference possible. The last question concerns the conciseness of the translation, and the usefulness of $\mathrm{ML^F}$: how many type annotations are needed? Actually, the translation removes type abstraction and type application, hence only type annotations on lambda-abstractions remain. Additionally, annotating with a monotype is useless in $\mathrm{ML^F}$, hence some of these annotations are unnecessary. In summary, we see that $\mathrm{ML^F}$ requires strictly less type information than System F. Moreover, all ML programs are typable without type annotations. Hence, we expect $\mathrm{ML^F}$ to be really usable in practice: it combines the conciseness of ML, and the second-order power of System F.

We first recall the definition of System F, then we give the translation of System F environments, as well as the translation of System F expressions. Finally, we prove the correctness of the translation.

9.1 Definition of System F

The types, terms, and typing contexts of system F are given below:

$$\begin{array}{rll} t ::= \alpha \mid t \to t \mid \forall\,\alpha \cdot t & & \text{Types} \\ M ::= x \mid M\ M' \mid \lambda(x:t)\ M \mid \Lambda(\alpha)\ M \mid M\ [t] & & \text{Terms} \\ A ::= \emptyset \mid A, x : t \mid A, \alpha & & \text{Typing Contexts} \end{array}$$

Types are not divided into monotypes and polytypes, as in $\mathrm{ML^F}$, and are simply polymorphic types: the $\forall$ quantifier can appear at any occurrence. Expressions are either variables such as x, applications $M\ M'$, annotated abstractions $\lambda(x:t)\ M$, type abstractions $\Lambda(\alpha)\ M$, which explicitly bind the type variable α in the expression M, or type applications $M\ [t]$, which explicitly instantiate the polymorphic type of M, say $\forall\,(\alpha)\ t'$, by using t, leading to $t'[t/\alpha]$. Typing environments bind variables to types, and bind free type variables. In System F, all type variables must be explicitly introduced.

The typing rules of System F are the following:

$$\textsc{F-Var}\quad \frac{x:t\in A}{A\vdash x:t}\qquad \textsc{F-App}\quad \frac{A\vdash a_1:t_2\to t_1\qquad A\vdash a_2:t_2}{A\vdash a_1\ a_2:t_1}\qquad \textsc{F-Tapp}\quad \frac{A\vdash a:\forall\,\alpha\cdot t}{A\vdash a\ [t']:t[t'/\alpha]}$$

$$\textsc{F-Fun}\quad \frac{A,x:t\vdash a:t'}{A\vdash \lambda(x:t)\ a:t\to t'}\qquad \textsc{F-Tfun}\quad \frac{A,\alpha\vdash a:t\qquad \alpha\notin A}{A\vdash \Lambda(\alpha)\ a:\forall\,\alpha\cdot t}$$

We see that polymorphism is explicitly introduced by type abstraction in Rule F-Tfun, and that it is explicitly instantiated by type application in Rule F-Tapp.

9.2 Encoding types and typing environments

The translation of types of System F into ML$^{\mathrm{F}}$ types uses auxiliary rigid bindings for arrow types. This ensures that there are no inner polytypes left in the result of the translation, which would otherwise be ill-formed. Quantifiers that are present in the original types are translated to unconstrained bounds.

$$[\![\alpha]\!]=\alpha\qquad [\![\forall\,\alpha\cdot t]\!]=\forall\,(\alpha)\ [\![t]\!]\qquad [\![t_1\to t_2]\!]=\forall\,(\alpha_1=[\![t_1]\!])\ \forall\,(\alpha_2=[\![t_2]\!])\ \alpha_1\to\alpha_2$$

In order to state the correspondence between typing judgments, we must also translate typing contexts. The translation of A, written $[\![A]\!]$, returns a pair $(Q)\ \Gamma$ of a prefix and a typing context and is defined inductively as follows:

$$[\![\emptyset]\!]=()\ \emptyset\qquad \frac{[\![A]\!]=(Q)\ \Gamma}{[\![A,x:t]\!]=(Q)\ \Gamma,x:[\![t]\!]}\qquad \frac{[\![A]\!]=(Q)\ \Gamma\qquad \alpha\notin\mathsf{dom}(Q)}{[\![A,\alpha]\!]=(Q,\alpha)\ \Gamma}$$

The translation of types enjoys the following property:

Lemma 9.2.1 *For any t and t', we have* $[\![t'[t/\alpha]]\!]\sqsubseteq\forall\,(\alpha=[\![t]\!])\ [\![t']\!]$.

Proof: Let σ be $[\![t]\!]$. We reason by structural induction on t'.

○ Case t' is α: The left-hand side is $[\![t]\!]$, that is σ, the right-hand side is $\forall\,(\alpha=\sigma)\ \alpha$. They are equivalent by Eq-Var. We conclude by A-Equiv.

∘ Case t' is β and $\beta \neq \alpha$: The left-hand side is β, the right-hand side is $\forall\,(\alpha = \sigma)\ \beta$. They are equivalent by Eq-Free, and we conclude by A-Equiv.

∘ Case t' is $\forall\,\beta \cdot t''$: By induction hypothesis, we have $\llbracket t''[t/\alpha] \rrbracket \sqsubseteq \forall\,(\alpha = \sigma)\ \llbracket t'' \rrbracket$. By notation, this means $(Q)\ \llbracket t''[t/\alpha] \rrbracket \sqsubseteq \forall\,(\alpha = \sigma)\ \llbracket t'' \rrbracket$, where Q is unconstrained and binds free variables of the judgment. We can freely assume that Q is of the form $(Q', \beta \geq \bot)$ (using Lemma 1.6.2 (page 57) if necessary). Hence, by R-Context-R, we get $\forall\,(\beta)\ \llbracket t''[t/\alpha] \rrbracket \sqsubseteq \forall\,(\beta)\ \forall\,(\alpha = \sigma)\ \llbracket t'' \rrbracket$ **(<u>1</u>)**. By definition, the left-hand side is $\llbracket \forall\,\beta \cdot t''[t/\alpha] \rrbracket$, *i.e.* $\llbracket t'[t/\alpha] \rrbracket$. The right-hand side is equivalent to $\forall\,(\alpha = \sigma)\ \forall\,(\beta)\ \llbracket t'' \rrbracket$ by Eq-Comm, *i.e.* $\forall\,(\alpha = \sigma)\ \llbracket t' \rrbracket$ by definition of $\llbracket \cdot \rrbracket$ and of t'. Hence, (1) gives $\llbracket t'[t/\alpha] \rrbracket \sqsubseteq \forall\,(\alpha = \sigma)\ \llbracket t' \rrbracket$, which is the expected result.

∘ Case t' is $t_1 \to t_2$: By induction hypothesis, we have $\llbracket t_1[t/\alpha] \rrbracket \sqsubseteq \forall\,(\alpha = \llbracket t \rrbracket)\ \llbracket t_1 \rrbracket$ and $\llbracket t_2[t/\alpha] \rrbracket \sqsubseteq \forall\,(\alpha = \llbracket t \rrbracket)\ \llbracket t_2 \rrbracket$. which we write $\sigma'_1 \sqsubseteq \forall\,(\alpha = \sigma)\ \sigma_1$ **(<u>2</u>)** and $\sigma'_2 \sqsubseteq \forall\,(\alpha = \sigma)\ \sigma_2$ **(<u>3</u>)**. We only have to show that $\forall\,(\alpha_1 = \sigma'_1)\ \forall\,(\alpha_2 = \sigma'_2)\ \alpha_1 \to \alpha_2 \sqsubseteq \forall\,(\alpha = \sigma)\ \forall\,(\alpha_1 = \sigma_1)\ \forall\,(\alpha_2 = \sigma_2)\ \alpha_1 \to \alpha_2$ holds. This is shown by R-Context-Rigid, (2) and (3), and by A-Up*. ■

9.3 Encoding expressions

The translation of System F terms into ML$^{\mathrm{F}}$ terms forgets type abstractions and applications, and translates types inside term abstractions.

$$\llbracket \Lambda(\alpha)\ M \rrbracket = \llbracket M \rrbracket \qquad \llbracket M\ t \rrbracket = \llbracket M \rrbracket \qquad \llbracket x \rrbracket = x \qquad \llbracket M\ M' \rrbracket = \llbracket M \rrbracket\ \llbracket M' \rrbracket$$

$$\llbracket \lambda(x : t)\ M \rrbracket = \lambda(x : \llbracket t \rrbracket)\ \llbracket M \rrbracket$$

The soundness of this translation is stated by the following theorem:

Theorem 5 *For any closed typing context A (that does not bind the same type variable twice), term M, and type t of System F such that $A \vdash M : t$, there exists a derivation $(Q)\ \Gamma \vdash \llbracket M \rrbracket : \sigma$ such that $(Q)\ \Gamma = \llbracket A \rrbracket$ and $\llbracket t \rrbracket \sqsubseteq \sigma$ hold.*

<u>Proof</u>: We reason by induction on the derivation of $A \vdash M : t$. Let $(Q)\ \Gamma$ be $\llbracket A \rrbracket$.

∘ Case F-Var: By hypothesis, we have $x : t \in A$. By definition of $\llbracket A \rrbracket$, we have $x : \llbracket t \rrbracket \in \Gamma$, and Γ is closed under Q. Hence, $(Q)\ \Gamma \vdash x : \llbracket t \rrbracket$ holds by Var. This is the expected result.

∘ Case F-App: We have $M = M_1\ M_2$ and the premises are $A \vdash M_1 : t_2 \to t_1$ and $A \vdash M_2 : t_2$. By induction hypothesis, we have both $(Q)\ \Gamma \vdash \llbracket M_1 \rrbracket : \sigma$ **(<u>1</u>)** and $(Q)\ \Gamma \vdash \llbracket M_2 \rrbracket : \sigma_2$ **(<u>2</u>)** with $\llbracket t_2 \to t_1 \rrbracket \sqsubseteq \sigma$ **(<u>3</u>)** and $\llbracket t_2 \rrbracket \sqsubseteq \sigma_2$ **(<u>4</u>)**. By definition of $\llbracket t_2 \to t_1 \rrbracket$, we have $\forall\,(\alpha_2 = \llbracket t_2 \rrbracket, \alpha_1 = \llbracket t_1 \rrbracket)\ \alpha_2 \to \alpha_1 \sqsubseteq \sigma$ **(<u>5</u>)** from (3). By (4) and R-Context-Rigid, we have $\forall\,(\alpha_2 = \llbracket t_2 \rrbracket, \alpha_1 = \llbracket t_1 \rrbracket)\ \alpha_2 \to \alpha_1 \sqsubseteq \forall\,(\alpha_2 = \sigma_2, \alpha_1 = \llbracket t_1 \rrbracket)\ \alpha_2 \to \alpha_1$ **(<u>6</u>)**. Thus by Lemma 2.8.2, (5), and (6), there exists a type σ' such that we have $\sigma \sqsubseteq \sigma'$ **(<u>7</u>)** and

$\forall\,(\alpha_2 = \sigma_2, \alpha_1 = \llbracket t_1 \rrbracket)\ \alpha_2 \to \alpha_1 \mathrel{\sqsubseteq\!\!\!\!-} \sigma'$ **(8)**. We can freely assume that σ' is in normal form. By Property 2.1.3.ii (page 65), σ' is of the form $\forall\,(Q')\ \tau_2 \to \tau_1$ **(9)**. By Inst, (1), (7), and (9), we get $(Q)\ \Gamma \vdash \llbracket M_1 \rrbracket : \forall\,(Q')\ \tau_2 \to \tau_1$ **(10)**. By Lemma 3.6.11, (8), and (9) we have $\sigma_2 \mathrel{\sqsubseteq\!\!\!\!-} \forall\,(Q')\ \tau_2$ **(11)** as well as $\llbracket t_1 \rrbracket \mathrel{\sqsubseteq\!\!\!\!-} \forall\,(Q')\ \tau_1$ **(12)**. Consequently, we can derive $(Q)\ \Gamma \vdash \llbracket M_2 \rrbracket : \forall\,(Q')\ \tau_2$ **(13)** by (11), (2), and Inst. Thus, $(Q)\ \Gamma \vdash \llbracket M_1 \rrbracket\ \llbracket M_2 \rrbracket : \forall\,(Q')\ \tau_1$ holds by App*, (10), and (13). This and (12) are the expected result.

○ Case F-Tapp: The premise is $A \vdash M : \forall\,\alpha \cdot t'$. By induction hypothesis, we have $(Q)\ \Gamma \vdash \llbracket M \rrbracket : \sigma$ **(14)** where $\llbracket \forall\,\alpha \cdot t' \rrbracket \mathrel{\sqsubseteq\!\!\!\!-} \sigma$ **(15)**. By definition, $\llbracket \forall\,\alpha \cdot t' \rrbracket$ is $\forall\,(\alpha)\ \llbracket t' \rrbracket$, thus (15) is $\forall\,(\alpha)\ \llbracket t' \rrbracket \mathrel{\sqsubseteq\!\!\!\!-} \sigma$. By Lemma 3.6.7, this implies that σ is equivalent to $\forall\,(\alpha)\ \sigma_0$ **(16)** with $\llbracket t' \rrbracket \mathrel{\sqsubseteq\!\!\!\!-} \sigma_0$ **(17)**. Let σ' be $\forall\,(\alpha = \llbracket t \rrbracket)\ \sigma_0$. We have $\sigma \sqsubseteq \sigma'$ **(18)** by (16), R-Context-Flexible, and I-Bot. By Rule Inst, (14), and (18) gives $(Q)\ \Gamma \vdash \llbracket M \rrbracket : \sigma'$. Furthermore, we have $\forall\,(\alpha = \llbracket t \rrbracket)\ \llbracket t' \rrbracket \mathrel{\sqsubseteq\!\!\!\!-} \sigma'$ **(19)** by (17) and R-Context-R. We note that $\llbracket t'[t/\alpha] \rrbracket \mathrel{\sqsubseteq\!\!\!\!-} \forall\,(\alpha = \llbracket t \rrbracket)\ \llbracket t' \rrbracket$ **(20)** holds by Lemma 9.2.1. Hence, by R-Trans, (20), and (19), we get $\llbracket t'[t/\alpha] \rrbracket \mathrel{\sqsubseteq\!\!\!\!-} \sigma'$. This is the expected result.

○ Case F-Fun: The premise is $A, x : t \vdash a : t'$. By induction, we have $(Q)\ \Gamma, x : \llbracket t \rrbracket \vdash a : \sigma'$, where $\llbracket t' \rrbracket \mathrel{\sqsubseteq\!\!\!\!-} \sigma'$ **(21)**. By Rule Fun*, we have $(Q)\ \Gamma \vdash \lambda(x : \llbracket t \rrbracket)\ a : \forall\,(\alpha = \llbracket t \rrbracket, \alpha' \geq \sigma')\ \alpha \to \alpha'$. Let σ'' be $\forall\,(\alpha = \llbracket t \rrbracket, \alpha' = \sigma')\ \alpha \to \alpha'$. By Rule Inst, we have $(Q)\ \Gamma \vdash a : \sigma''$. By (21), we have: $\forall\,(\alpha = \llbracket t \rrbracket, \alpha' = \llbracket t' \rrbracket)\ \alpha \to \alpha' \mathrel{\sqsubseteq\!\!\!\!-} \sigma''$, *i.e.* $\llbracket t \to t' \rrbracket \mathrel{\sqsubseteq\!\!\!\!-} \sigma''$.

○ Case F-Tfun: The premise is $A, \alpha \vdash a : t$ By induction, we have $(Q, \alpha)\ \Gamma \vdash a : \sigma$ with $\llbracket t \rrbracket \mathrel{\sqsubseteq\!\!\!\!-} \sigma$ **(22)**. We have $\alpha \notin \Gamma$, since $\alpha \notin A$. By Rule Gen, we have $(Q)\ \Gamma \vdash a : \forall\,(\alpha)\ \sigma$. From (22), we have $\forall\,(\alpha)\ \llbracket t \rrbracket \mathrel{\sqsubseteq\!\!\!\!-} \forall\,(\alpha)\ \sigma$. That is $\llbracket \forall\,\alpha \cdot t \rrbracket \mathrel{\sqsubseteq\!\!\!\!-} \forall\,(\alpha)\ \sigma$. ■

Noticeably, translated terms contain strictly fewer annotations than original terms; this property that was not true in Poly-ML. In particular, all type Λ-abstractions and type applications are dropped and only annotations of λ-bound variables remain. Moreover, some of these annotations are still superfluous, since, for instance, removing monotype annotations preserves typings in ML$^{\mathrm{F}}$.

Remark 9.3.1 System F can be viewed in Curry style, where terms are unannotated, or in Church style, where terms come with sufficient type information so that full type information can be uniquely reconstructed. In Church style, terms can be given a typed semantics. For instance, type abstraction may stop the evaluation just like value abstractions.

UML$^{\mathrm{F}}$ is somehow the Curry's view. However, there is no exact Church style for ML$^{\mathrm{F}}$: type abstractions, *i.e.* places where polymorphism is introduced, are left implicit, and type applications, *i.e.* places where polymorphism is used, are inferred. Even terms with fully annotated λ-abstractions can place type-abstraction and type application at different occurrences.

There is thus no obvious typed semantics for ML$^{\mathrm{F}}$: if the semantics were given on type derivations, one should then show some coherence property that the semantics is independent of the translation, and in particular on places where type-abstractions or type-applications would be inserted.

Encoding Poly-ML

Although not stated formally, terms of Poly-ML [GR99] can be translated to terms of System F. As a corollary of Theorem 5, terms typable in Poly-ML can also be typed in ML^{F}. note that all polymorphism-elimination annotations $\langle\cdot\rangle$ that were mandatory in Poly-ML are indeed removed in the translation.

Chapter 10

Shallow ML^{F}

We have seen in Chapter 9 that System F can be encoded in ML^{F} by just erasing type applications and type abstractions. Hence, the set of unannotated lambda-terms typable in System F is included in the set of lambda-terms typable in ML^{F}. Conversely, we may wonder whether ML^{F} is strictly more powerful than System F, that is, whether there exists a term typable in ML^{F} whose type erasure is a lambda-term that is not typable in System F. As a first step, we identify a strict subset of ML^{F} which is sufficient to encode System F. In this chapter, we describe this subsystem, called Shallow ML^{F}.

10.1 Definition and characterization

In the following, a type of System F is called a System F type. We also introduce a subset of ML^{F} types, called *F-types*, which are, intuitively, the translation of System F types into ML^{F}.

Definition 10.1.1 A type in normal form is an *F-type* if and only if all its flexible bounds are unconstrained (that is, of the form $(\alpha \geq \bot)$). A type is an *F-type* if and only if its normal form is an F-type. □

Types that are not F-types, *e.g.* $\forall\,(\alpha \geq \sigma)\ \tau$ $(\underline{\mathbf{1}})$, where σ is equivalent neither to a monotype nor to $\bot$, have been introduced to factor out choices during type inference. Such types are indeed used in a derivation of `let` $f =$ `choose id in` $(f$ `auto`$)$ $(f$ `succ`$)$. However, they are never used in the encoding of System F. We wish to show that restricting ML^{F} derivations to use only F-types preserves the encoding of System F. We do not show this result directly, but first consider a less restrictive subset of types, called *shallow* types. Shallow types are a superset of F-types that also allow types such as (1), except in rigid bounds. Indeed, a rigid bound corresponds intuitively to an explicit type annotation in the source code. Then restricting rigid bounds to F-types amounts to restricting type annotations to F-types only. From a practical point of view, this means that the type annotation primitives available to the programmer are F-types only. Shallow types are still needed, however, because they are introduced by the type inference algorithm *e.g.* in the typing of `choose id`).

Definition 10.1.2 Let a type in normal form be *shallow* if and only if all its rigid bounds are F-types. A type is *shallow* if and only if its normal form is shallow. A prefix Q is shallow if and only if $\forall\,(Q)\;\nabla_{\mathsf{dom}(Q)}$ is shallow. □

Since shallowness depends on the binding structure of types, it can be captured by polynomials, as defined in Section 2.7.2. Actually, polynomial variables X, Y, and Z correspond respectively to top-level flexible bindings, rigid bindings, and flexible bindings under rigid bindings. Hence, we expect Z to correspond exactly to the bounds that are unconstrained in shallow types. which leads to the following definition:

Definition 10.1.3 Let P and P' be polynomials in $\mathbb{N}[X,Y,Z]$. We write $P \leqslant_Z P'$ if and only if the maximal degree of variable Z in P is less than or equal to the maximal degree of variable Z in P'. □

For example, we have $X^2YZ \leqslant_Z Z^2$, $XY \leqslant_Z Y$, as well as $1 \leqslant_Z 0$. Note that $\leqslant_Z$ is defined with the maximal degree, hence we have some unusual properties. For instance we have $P_1 + P_2 \leqslant_Z P'$ if and only if $P_1 \leqslant_Z P'$ *and* $P_2 \leqslant_Z P'$. Moreover, we have $P \leqslant_Z P_1 + P_2$ if and only if $P \leqslant_Z P_1$ *or* $P \leqslant_Z P_2$.

The shallowness of a type σ can be tested by looking only at the degree of its weight in variable Z (weights w_A were defined page 91), as formalized by Property 10.1.4.ii below. For example, the type $\forall\,(\alpha \geq \sigma_{\mathtt{id}}, \beta = \sigma_{\mathtt{id}})\;\alpha \to \beta$ is shallow, and its polynomial is $X^2 + YZ$. Conversely, the type $\forall\,(\beta = \forall\,(\alpha \geq \sigma_{\mathtt{id}})\;\alpha \to \alpha)\;\beta \to \beta$ is not shallow, and its polynomial is YZ^2.

Properties 10.1.4

i) *A type σ is an F-type iff $w_Y(\sigma) \leqslant_Z Z$.*

ii) *A type σ is shallow iff $w_X(\sigma) \leqslant_Z Z$.*

iii) *For any σ, we have $w_X(\sigma) \leqslant_Z w_Y(\sigma)$.*

Proof: Property i: By Property 1.5.6.i and Lemma 2.7.5, we have $w_Y(\sigma) = w_Y(\mathsf{nf}(\sigma))$. Moreover, by definition, a type is shallow if and only if its normal form is shallow. As a consequence, it suffices to show that $\mathsf{nf}(\sigma)$ is an F-type if and only if $w_Y(\mathsf{nf}(\sigma)) \leqslant_Z Z$. To ease the presentation, we simply assume that σ is in normal form. We proceed by case analysis on the form of σ.

- Case τ or $\bot$: Then $w_Y(\sigma)$ is 0 or 1, and $w_Y(\sigma) \leqslant_Z Z$ holds.

- Case $\forall\,(\alpha \geq \sigma_1)\;\sigma_2$: Then $w_Y(\sigma)$ is $w_Z(\sigma_1) \times Z + w_Y(\sigma_2)$ $\underline{(\mathbf{1})}$. If σ is an F-type, then, by definition, σ_1 is $\bot$, and σ_2 is an F-type. Hence, we have $w_Z(\sigma_1) = 1$ by definition and

$w_Y(\sigma_2) \leqslant_Z Z$ **(2)** by induction hypothesis. This gives $w_Z(\sigma_1) \times Z \leqslant_Z Z$, which implies $w_Y(\sigma) \leqslant_Z Z$ by (1) and (2). Conversely, if $w_Y(\sigma) \leqslant_Z Z$ holds, then from (1), we have $w_Z(\sigma_1) \leqslant_Z 1$ and $w_Y(\sigma_2) \leqslant_Z Z$. The former implies that $w_Z(\sigma_1)$ is 0 or 1. Hence, σ_1 is $\bot$ (it cannot be a monotype τ because σ is in normal form). The latter implies, by induction hypothesis, that σ_2 is an F-type. Hence, σ, that is $\forall\,(\alpha \geq \bot)\ \sigma_2$, is an F-type.

○ CASE $\forall\,(\alpha = \sigma_1)\ \sigma_2$: Then $w_Y(\sigma)$ is $w_Y(\sigma_1) \times Y + w_Y(\sigma_2)$. Hence, on the one hand, we have $w_Y(\sigma) \leqslant_Z Z$ if and only if $w_Y(\sigma_1) \leqslant_Z Z$ and $w_Y(\sigma_2) \leqslant_Z Z$. On the other hand, σ is an F-type if and only if σ_1 and σ_2 are F-types. We conclude by induction hypothesis.

Property ii: By structural induction on σ. As in Property i, we assume that σ is in normal form.

○ CASE τ or $\bot$: Then $w_X(\sigma)$ is 0 or 1, and $w_X(\sigma) \leqslant_Z Z$ holds.

○ CASE $\forall\,(\alpha \geq \sigma_1)\ \sigma_2$: Then $w_X(\sigma)$ is $w_X(\sigma_1) \times X + w_X(\sigma_2)$. Hence, on the one hand, we have $w_X(\sigma) \leqslant_Z Z$ if and only if $w_X(\sigma_1) \leqslant_Z Z$ and $w_X(\sigma_2) \leqslant_Z Z$. On the other hand, σ is shallow if and only if σ_1 and σ_2 are shallow. We conclude by induction hypothesis.

○ CASE $\forall\,(\alpha = \sigma_1)\ \sigma_2$: Then $w_X(\sigma)$ is $w_Y(\sigma_1) \times Y + w_X(\sigma_2)$. Hence, we have $w_X(\sigma) \leqslant_Z Z$ if and only if $w_Y(\sigma_1) \leqslant_Z Z$ and $w_X(\sigma_2) \leqslant_Z Z$. We conclude by induction hypothesis and Property i.

Property iii: By structural induction on σ. ■

These properties provide a convenient way to characterize F-types and shallow types. It remains to define Shallow ML^{F} as a subset of ML^{F}: Recall that ML^{F} (with type annotations) is a type system that allows all ML^{F}-types as annotations and in derivations. Additionally, $\text{ML}^{\text{F}}_{\star}$ allows the guessing of type annotations thanks to Rule ORACLE.

Definition 10.1.5 (Shallow ML^{F}) We define Shallow ML^{F} as the subset of ML^{F} where type annotations are restricted to F-types and derivations can only mention shallow types. Similarly, Shallow $\text{ML}^{\text{F}}_{\star}$ is the subset of $\text{ML}^{\text{F}}_{\star}$ where derivations can only mention shallow types and Rule ORACLE can only be used with F-types. □

As mentioned earlier, restricting type annotations to F-types amounts to restricting the type of the annotation primitive to a shallow type. Indeed, the type annotation primitive (σ) (we assume σ is closed) has type $\forall\,(\alpha_1 = \sigma)\ \forall\,(\alpha_2 = \sigma)\ \alpha_1 \to \alpha_2$. This type is shallow if and only if σ is an F-type.

We have given an encoding of System F into ML^{F} in Chapter 9. In the next section, we show that this encoding is actually an encoding into Shallow ML^{F}. That is, not only the type annotations are F-types (which is obvious), but the typing derivations themselves are actually shallow. This means that Shallow ML^{F}, though it is a subset of ML^{F}, is sufficient to provide all the expressiveness of System F.

10.2 Expressiveness of Shallow ML$^{\textsf{F}}$

It is immediate to check that the translation of System F types are F-types. Hence, the encoding of a term of System F gives a term whose type annotations are F types. Besides, the encoding of a typing environment is a pair composed of a shallow typing environment and an unconstrained prefix. As a consequence, we expect the typing derivation of the encoding to be shallow. The main result of this section gives a stronger result (it is not restricted to translations of System F terms): Lemma 10.2.3 states that if a term is typable in ML$^{\textsf{F}}$ under a shallow typing environment and a shallow prefix, then it is typable in Shallow ML$^{\textsf{F}}$. The next lemma is a preliminary result which implies that an abstraction of an F-type is also an F-type.

Lemma 10.2.1 *If $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ holds, then so does $w_Y(\sigma_2) \leqslant_Z w_Y(\sigma_1)$.*

See proof in the Appendix (page 296).

Our goal is to show that a shallow typable term admits a shallow derivation. Actually, the inference and unification algorithms are “shallow-stable”, that is, when given shallow terms (resp. types), they return shallow derivations (resp. types). This implies that the principal type of a shallow term is a shallow type. We formalize the above result for the unification algorithm, after having stated two useful properties.

Properties 10.2.2

i) *If Q is shallow, then $\forall\,(Q)\ \tau$ is shallow.*

ii) *If Q is shallow and $(Q_1, Q_2) = Q{\uparrow}\bar{\alpha}$, then Q_1 and Q_2 are shallow.*

iii) *If Q is shallow and $Q' =$ `unify` (Q, τ_1, τ_2), then Q' is shallow.*

iv) *If Q, σ_1, and σ_2 are shallow, and $(Q', \sigma_3) =$ `polyunify` (Q, σ_1, σ_2), then Q' and σ_3 are shallow.*

Proof: Properties i and ii are easily shown by structural induction on Q. We prove properties iii and iv simultaneously by induction on the recursive calls to the algorithms. All cases are easy, except the case `unify` (Q, α_1, α_2), where $(\alpha_1 \diamond_1 \sigma_1)$ and $(\alpha_2 \diamond_2 \sigma_2)$ are in Q. In this case, we call `polyunify` as follows: $(Q', \sigma_3) =$ `polyunify` (Q, σ_1, σ_2). By induction hypothesis Q' and σ_3 are shallow. Besides, without loss of generality, Q' is of the form $(Q_1, \alpha_1 \diamond_1 \sigma_1, Q_2, \alpha_2 \diamond_2 \sigma_2, Q_3)$. We note that Q_1, Q_2, and Q_3 are shallow. We return Q'' defined as $(Q_1, \alpha_1 \diamond \sigma_3, Q_2, \alpha_2 = \alpha_1, Q_3)$, where $\diamond$ is rigid if and only if $\diamond_1$ or $\diamond_2$ is rigid. If $\diamond$ is flexible, then Q'' is shallow by construction, and the case is solved. Otherwise, $\diamond$ is rigid, which means that $\diamond_1$ or $\diamond_2$ is rigid. We show the result for $\diamond_1$ rigid, the other case being similar. The update algorithm calls the abstraction-check algorithm, which ensures that $(Q')\ \sigma_1 \sqsubseteq \sigma_3$ holds. By hypothesis, $(\alpha_1 = \sigma_1)$ is in Q and Q is shallow, thus σ_1 must be an F-type. By Lemma 10.2.1 and Property 10.1.4.i, σ_3 is an F-type too. Hence, Q'' is shallow. ■

We can now state the main result of the section: a shallow term typable in ML$^{\mathrm{F}}$ is typable in Shallow ML$^{\mathrm{F}}$.

Lemma 10.2.3 *If Q, Γ, and a are shallow, and $(Q)\ \Gamma \vdash a : \sigma$ holds in ML^{F} for some σ, then there exists a shallow derivation of $(Q)\ \Gamma \vdash a : \sigma'$ for some shallow type σ'.*

Proof: By completeness of the type inference algorithm, we know that `infer` (Q, Γ, a) succeeds. We show by induction on the recursive calls to `infer` that if Q, Γ, and a are shallow, and Q', σ is `infer` (Q, Γ, a), then Q' and σ are shallow. All cases are easy, using Property 10.2.2.iii for the application case. ■

As a conclusion, if a shallow term a is typable in ML$^{\mathrm{F}}$, under a shallow prefix and a shallow typing environment, then it admits a derivation in Shallow ML$^{\mathrm{F}}$. In that case, it is noticeable that the inference algorithm returns a shallow principal type. As a first corollary, subject reduction holds in Shallow ML$^{\mathrm{F}}$ since it holds in ML$^{\mathrm{F}}$. As a second corollary, the encoding given for System F also holds in Shallow ML$^{\mathrm{F}}$. Hence we have shown the inclusion System F $\subseteq$ Shallow ML$^{\mathrm{F}}$ (considering only type erasure of terms). The next section covers the converse inclusion: is Shallow ML$^{\mathrm{F}}$ included in System F?

10.3 Comparison with System F

10.3.1 Introduction

An immediate issue prevents Shallow ML$^{\mathrm{F}}$ from being compared to System F: Let-bindings. Usually, a let-binding `let` $x = a_1$ `in` a_2 in encoded as $(\lambda(x)\ a_2)\ a_1$. We note that let-bindings do not increase expressiveness in ML$^{\mathrm{F}}$ or ML$^{\mathrm{F}}_{\star}$, since the above encoding can always be used, inserting explicit type annotations or oracles if necessary. For example, let σ be the type $\forall\,(\alpha \geq \sigma_{\mathtt{id}})\ \alpha \rightarrow \alpha$. Then the expression `let` $x =$ `choose id in` a can be encoded as $(\lambda(x:\sigma)\ a)$ (`choose id`) in ML$^{\mathrm{F}}$, or as $(\lambda(x:\star)\ a)$ (`choose id`) in ML$^{\mathrm{F}}_{\star}$. The same property is not true for Shallow ML$^{\mathrm{F}}$, since shallow-types that are not F-types cannot be used as annotations. Indeed, the principal type given to `choose id`, namely σ, is a shallow type but not a F-type. Hence, whereas `let` $x =$ `choose id in` a is in Shallow ML$^{\mathrm{F}}$, its natural encoding as a λ-abstraction is not in Shallow ML$^{\mathrm{F}}$.[1] Therefore, we also have to consider the restriction Shallow F of Shallow ML$^{\mathrm{F}}$ to programs without let-bindings. The encoding of System F into ML$^{\mathrm{F}}$ given in Section 9.1 is actually an encoding into Shallow F, by considering $\lambda(x:\sigma)\ a$ as syntactic sugar for $\lambda(x)\ a[(x:\sigma)/x]$ instead of $\lambda(x)$ `let` $x = (x:\sigma)$ `in` a.

---

[1] This does not imply that the type erasure of the encoding is not in Shallow ML$^{\mathrm{F}}$. See the comparison between ML$^{\mathrm{F}}$ and System F in Section 10.4 for more details about this issue.

Similarly, $\lambda(x:\star)\ a$ is syntactic sugar for $\lambda(x)\ a[(x:\star)/x]$. Hence, we have the inclusion System F $\subseteq$ Shallow F, considering only the type erasure of terms. We wish to show the converse inclusion: the type erasure of terms typable in Shallow F, that is in Shallow ML^{F} without Let-binding, is included in the type erasure of terms typable in System F.

10.3.2 Preliminary results about System F

Types of System F are written with letter t. We recall that types of Shallow ML^{F} are written σ in general, and τ when they are monotypes. We implicitly consider that a monotype τ is in Shallow ML^{F} as well as in System F. We define below the projection of an F-type (which is a type in ML^{F}) into a System F type: Given an F-type σ in normal form, we define $\underline{\sigma}$ as its projection into an F-type:

$$\underline{\bot} \triangleq \forall\,\alpha\cdot\alpha \qquad \underline{\tau} \triangleq \tau \qquad \underline{\forall\,(\alpha=\sigma)\ \sigma'} \triangleq \underline{\sigma'}[\underline{\sigma}/\alpha] \qquad \underline{\forall\,(\alpha\geq\bot)\ \sigma} \triangleq \forall\,\alpha\cdot\underline{\sigma}$$

As for the general case, $\underline{\sigma}$ is $\underline{\mathsf{nf}(\sigma)}$. As mentioned above, we consider that τ is in System F, thus we have $\underline{\tau}=\tau$.

Next, we formalize the instance relation of System F, which we write $\sqsubseteq_F$. We use it to define the function $\mathsf{fsub}\,(t)$ as the set of all instances of t. The relation $\sqsubseteq_F$ is defined as follows:

$$\forall\,\bar{\alpha}\cdot t \sqsubseteq_F \forall\,\bar{\beta}\cdot t[\bar{t}/\bar{\alpha}] \text{ holds if and only if } \bar{\beta} \text{ is disjoint from } \mathsf{ftv}(t).$$

We define $\mathsf{fsub}\,(t)$ as $\{t' \mid t \sqsubseteq_F t'\}$.

We also extend the function $\mathsf{proj}()$ (defined page 40) to System F types. This function takes a System F type and returns a skeleton.

$$\mathsf{proj}(\alpha) \triangleq \alpha \qquad \mathsf{proj}(t_1 \to t_2) \triangleq \mathsf{proj}(t_1) \to \mathsf{proj}(t_2) \qquad \mathsf{proj}(\forall\,\alpha\cdot t) \triangleq \mathsf{proj}(t)[\bot/\alpha]$$

We write $\sigma \leqslant_/ t$ for $\sigma/ \leqslant_/ t/$.

We can now state a few properties. In the following, we write S for a set of types of System F. We define $\forall\,\alpha\cdot S$ as the set $\{\forall\,\alpha\cdot t \mid t\in S\}$, and $\mathsf{fsub}\,(S)$ is the set $\bigcup_{t\in S}\mathsf{fsub}\,(t)$. We use the letter θ for substitutions in System F. It should be clear from the context if a given substitution θ is a substitution in System F or in Shallow ML^{F}. Substitutions are implicitly capture-avoiding. We note that $[(\forall\,\beta\cdot\beta\to\beta)/\alpha]$ is a valid substitution in System F. It is not the case in Shallow ML^{F} or in ML^{F}, where only monotype substitutions are allowed. We write $\theta(S)$ for the set $\{\theta(t) \mid t\in S\}$.

Properties 10.3.1

i) *We have* $\mathsf{ftv}(\underline{\sigma}) = \mathsf{ftv}(\sigma)$.

ii) *We have* $\theta(\mathsf{fsub}\,(S)) \subseteq \mathsf{fsub}\,(\theta(S))$.

iii) *We have* $\forall\,\alpha \cdot \mathsf{fsub}\,(S) \subseteq \mathsf{fsub}\,(\forall\,\alpha \cdot S)$.

iv) *If* $\mathsf{fsub}\,(S_1) = \mathsf{fsub}\,(S_2)$*, then* $\mathsf{fsub}\,(\forall\,\bar{\alpha} \cdot S_1) = \mathsf{fsub}\,(\forall\,\bar{\alpha} \cdot S_2)$.

v) *We have* $\mathsf{fsub}\,(\theta(t)) = \mathsf{fsub}\,(\theta(\mathsf{fsub}\,(t)))$.

Proof: Property i: Straightforward structural induction on σ.

Property ii: Let t be in $\theta(\mathsf{fsub}\,(S))$. By definition, there exists $t_1 \in S$ and t_2 such that $t_1 \sqsubseteq_F t_2$ **(1)** and $t = \theta(t_2)$. Then we get $\theta(t_1) \sqsubseteq_F \theta(t_2)$ from (1), that is, $\theta(t_1) \sqsubseteq_F t$. Hence, $t \in \mathsf{fsub}\,(\theta(S))$.

Property iii: Let t be in $\forall\,\alpha \cdot \mathsf{fsub}\,(S)$. By definition, t is of the form $\forall\,\alpha \cdot t'$ for some System F type t' and there exists t'' in S such that $t'' \sqsubseteq_F t'$ **(1)**. Then $\forall\,\alpha \cdot t'' \sqsubseteq_F \forall\,\alpha \cdot t'$ holds from (1), that is $\forall\,\alpha \cdot t'' \sqsubseteq_F t$. Hence, $t \in \mathsf{fsub}\,(\forall\,\alpha \cdot S)$.

Property iv: We have $\mathsf{fsub}\,(S_1) \subseteq \mathsf{fsub}\,(S_2)$. Prefixing both sets with $\forall\alpha$, we get the inclusion $\forall\,\alpha \cdot \mathsf{fsub}\,(S_1) \subseteq \forall\,\alpha \cdot \mathsf{fsub}\,(S_2)$. Applying $\mathsf{fsub}\,()$, we get $\mathsf{fsub}\,(\forall\,\alpha \cdot \mathsf{fsub}\,(S_1)) \subseteq \mathsf{fsub}\,(\forall\,\alpha \cdot \mathsf{fsub}\,(S_2))$ **(1)**. By Property iii, we have $\forall\,\alpha \cdot \mathsf{fsub}\,(S_2) \subseteq \mathsf{fsub}\,(\forall\,\alpha \cdot S_2)$. Hence, we get $\mathsf{fsub}\,(\forall\,\alpha \cdot \mathsf{fsub}\,(S_1)) \subseteq \mathsf{fsub}\,(\mathsf{fsub}\,(\forall\,\alpha \cdot S_2))$ from (1). This happens to be equivalent to $\mathsf{fsub}\,(\forall\,\alpha \cdot \mathsf{fsub}\,(S_1)) \subseteq \mathsf{fsub}\,(\forall\,\alpha \cdot S_2)$ **(2)**. Additionally, we have $S_1 \subseteq \mathsf{fsub}\,(S_1)$, thus $\mathsf{fsub}\,(\forall\,\alpha \cdot S_1) \subseteq \mathsf{fsub}\,(\forall\,\alpha \cdot \mathsf{fsub}\,(S_1))$ holds. Then, $\mathsf{fsub}\,(\forall\,\alpha \cdot S_1) \subseteq \mathsf{fsub}\,(\forall\,\alpha \cdot S_2)$ holds from (2). By symmetry, we get $\mathsf{fsub}\,(\forall\,\alpha \cdot S_1) = \mathsf{fsub}\,(\forall\,\alpha \cdot S_2)$.

Property v: We have $t \in \mathsf{fsub}\,(t)$. Hence, $\theta(t)$ is in the set $\theta(\mathsf{fsub}\,(t))$, which implies that $\mathsf{fsub}\,(\theta(t)) \subseteq \mathsf{fsub}\,(\theta(\mathsf{fsub}\,(t)))$ **(1)** holds. Conversely, we show that $\mathsf{fsub}\,(\theta(\mathsf{fsub}\,(t))) \subseteq \mathsf{fsub}\,(\theta(t))$ holds. Let t_1 be in $\mathsf{fsub}\,(\theta(\mathsf{fsub}\,(t)))$. By definition, there exists t_2 such that $t \sqsubseteq_F t_2$ **(2)** and $\theta(t_2) \sqsubseteq_F t_1$ **(3)**. Then $\theta(t) \sqsubseteq_F \theta(t_2)$ **(4)** holds from (2). Thus $\theta(t) \sqsubseteq_F t_1$ holds by (4) and (3). This means that t_1 is in $\mathsf{fsub}\,(\theta(t))$. As a conclusion, we have $\mathsf{fsub}\,(\theta(\mathsf{fsub}\,(t))) \subseteq \mathsf{fsub}\,(\theta(t))$ **(5)**. By (1) and (5), we get $\mathsf{fsub}\,(\theta(t)) = \mathsf{fsub}\,(\theta(\mathsf{fsub}\,(t)))$. ■

The next definition captures the idea that a set S depends only on a given set of type variables. For example, given a type t, consider the set S defined as $\mathsf{fsub}\,(t)$. Since S is generated only from t, it can depend only on the free variables of t. As a consequence, if the type $\beta \to \beta$ is in S, and β is not free in t, this means that $\gamma \to \gamma$ is also in S for any γ. Actually, we can even expect $t' \to t'$ to be in S, for any type t'. This is captured by the following definition (think of $\bar{\alpha}$ as the set of type variables used to generate the set S):

Definition 10.3.2 We say that a set of types S is $\bar{\alpha}$-stable if and only if for any substitution θ invariant on $\bar{\alpha}$, we have $\theta(S) \subseteq S$. □

We note that if S is $\bar{\alpha}$-stable, then it is also $(\bar{\alpha} \cup \bar{\beta})$-stable. We defined $\Sigma_{\bar{\alpha}}$ page 103 as the set of ML^{F}-types whose unbound variables are included in $\bar{\alpha}$. We now define $\Sigma_{\bar{\alpha}}^{\neg}$ as the set of System F types whose free variables are disjoint from $\bar{\alpha}$. The notation $S_1 =_{\bar{\alpha}} S_2$, defined next, intuitively means that the sets S_1 and S_2 are equal, except for some "uninteresting" types that mentions variables in $\bar{\alpha}$. Formally, we write $S_1 =_{\bar{\alpha}} S_2$ if $S_1 \cap \Sigma_{\bar{\alpha}}^{\neg} = S_2 \cap \Sigma_{\bar{\alpha}}^{\neg}$ holds. We write $S_1 \subseteq_{\bar{\alpha}} S_2$ if $S_1 \cap \Sigma_{\bar{\alpha}}^{\neg} \subseteq S_2 \cap \Sigma_{\bar{\alpha}}^{\neg}$ holds. We note that $S_1 =_{\bar{\alpha}} S_2$ holds if and only if $S_1 \subseteq_{\bar{\alpha}} S_2$ and $S_1 \subseteq_{\bar{\alpha}} S_2$ hold. Additionally, $S_1 =_{\emptyset} S_2$ means $S_1 = S_2$.

Properties 10.3.3

i) *If S is $\bar{\alpha}$-stable, then* $\mathsf{fsub}\,(S)$ *is $\bar{\alpha}$-stable.*
ii) *The set* $\mathsf{fsub}\,(t)$ *is* $\mathsf{ftv}(t)$*-stable.*
iii) *If S is $\bar{\alpha}$-stable and $\bar{\beta}$-stable, then it is $(\bar{\alpha} \cap \bar{\beta})$-stable.*
iv) *If S_1 and S_2 are $\bar{\alpha}$-stable, $\bar{\alpha} \# \bar{\beta}$, and $S_1 \subseteq_{\bar{\beta}} S_2$, then $S_1 \subseteq S_2$.*
v) *If S_1 and S_2 are $\bar{\alpha}$-stable, $\bar{\alpha} \# \bar{\beta}$, and $S_1 =_{\bar{\beta}} S_2$, then $S_1 = S_2$.*
vi) *If we have $S_1 \subseteq_{\bar{\alpha} \cup \{\alpha\}} S_2$ for all α in an infinite set I, then $S_1 \subseteq_{\bar{\alpha}} S_2$ holds.*
vii) *If we have $S_1 =_{\bar{\alpha} \cup \{\alpha\}} S_2$ for all α in an infinite set I, then $S_1 =_{\bar{\alpha}} S_2$ holds.*
viii) *If we have $S_1 \subseteq_{\bar{\alpha}} S_2$, then* $\mathsf{fsub}\,(S_1) \subseteq_{\bar{\alpha}} \mathsf{fsub}\,(S_2)$ *holds.*
ix) *If we have $S_1 =_{\bar{\alpha}} S_2$, then* $\mathsf{fsub}\,(S_1) =_{\bar{\alpha}} \mathsf{fsub}\,(S_2)$ *holds.*
x) *If we have $\bar{\alpha} \# \mathsf{codom}(\theta)$ and $S_1 =_{\bar{\alpha}} S_2$, then $\theta(S_1) =_{\bar{\alpha}} \theta(S_2)$.*

See proof in the Appendix (page 297).

Thanks to these properties about sets of System F types, we are ready to introduce the interpretation of Shallow ML^{F} types as sets of System F types.

10.3.3 Encoding shallow types into System F

Shallow types are more expressive than types of System F. As an example, the principal type given to the expression `choose id`, that is, $\forall\,(\alpha \geq \sigma_{\mathtt{id}})\ \alpha \rightarrow \alpha$ has no direct counterpart in System F. More precisely, the above expression can be annotated in multiple ways, leading to multiple typings in System F. In particular, it can be typed with $\sigma_{\mathtt{id}} \rightarrow \sigma_{\mathtt{id}}$ and also with $(t \rightarrow t) \rightarrow (t \rightarrow t)$ for any type t. This is the reason why the encoding of a shallow type is not a single System F type, but rather a set of types. Intuitively, the encoding of a type σ is the set of all instances of σ that are F-types. Hence, the following definition:

Definition 10.3.4 The interpretation of a shallow type σ, written $(\!(\sigma)\!)$, is a set of System F types defined inductively as follows:

$$(\!(\tau)\!) \triangleq \mathsf{fsub}\,(\tau) \qquad (\!(\bot)\!) \triangleq \mathsf{fsub}\,(\forall\,\alpha\cdot\alpha) \qquad (\!(\forall\,(\alpha=\sigma)\;\sigma')\!) \triangleq \mathsf{fsub}\left((\!(\sigma')\!)[\underline{\sigma}/\alpha]\right) \quad (\underline{\mathbf{1}})$$

$$(\!(\forall\,(\alpha\geq\sigma)\;\sigma')\!) \triangleq \mathsf{fsub}\left(\bigcup_{t\in(\!(\sigma)\!)}^{\bar\alpha\#\mathsf{ftv}(\sigma,\sigma')} \forall\,\bar\alpha\cdot(\!(\sigma')\!)[t/\alpha]\right)$$

□

We note that $(\!(\tau)\!)$ is $\mathsf{fsub}\,(\tau)$, which is different from $\{\tau\}$ since it contains all types equivalent to τ is System F. For instance, the type $\forall\,\alpha\cdot\tau$ is in $\mathsf{fsub}\,(\tau)$ if $\alpha\notin\mathsf{ftv}(\tau)$. In (1), note that σ is an F-type since $\forall\,(\alpha=\sigma)\;\sigma'$ is shallow. Hence, we substitute α by $\underline{\sigma}$ in the set $(\!(\sigma')\!)$.

Example The interpretation of the polymorphic type $\sigma_{\mathtt{id}}$ is (after simplification) $\mathsf{fsub}\left(\bigcup_{t}^{\bar\alpha\#\{\alpha\}} \forall\,\bar\alpha\cdot t\to t\right)$, which is exactly the set $\mathsf{fsub}\,(\forall\,\alpha\cdot\alpha\to\alpha)$. Actually, we show in Property 10.3.6.viii, below, that the interpretation of any F-type σ is the set of all instances of $\underline{\sigma}$ in System F. Note also that we consider the union over all $\bar\alpha$ disjoint from $\{\alpha\}$. This restriction is unnecessary, and we could as well consider variables $\bar\alpha$ containing α. However, since both definitions are equivalent, we chose the first one, which makes some proofs easier. It will be shown in Property 10.3.6.ix that more restrictions can be put on $\bar\alpha$ without changing the definition.

Given two sets of substitutions S_1 and S_2, we write $S_1 \circledcirc S_2$ for the set $\{\theta_1\circ\theta_2 \mid \theta_1\in S_1, \theta_2\in S_2\}$. We also write $S_1\circ\theta$ for $S_1\circledcirc\{\theta\}$.

Definition 10.3.5 The interpretation of a shallow prefix Q, written $(\!(Q)\!)$, is a set of substitutions of System F, defined inductively as follows:

$$(\!(\emptyset)\!) \triangleq \{\mathtt{id}\} \qquad\qquad (\!(Q',\alpha=\sigma)\!) \triangleq (\!(Q')\!)\circ[\underline{\sigma}/\alpha]$$

$$(\!(Q',\alpha\geq\sigma)\!) \triangleq (\!(Q')\!) \circledcirc \bigcup_{t\in(\!(\sigma)\!)} [\mathsf{nf}(t)/\alpha] \;(\underline{\mathbf{2}})$$

□

In (2), we write $\mathsf{nf}(t)$ to mean the normal form of t in System F. We could equivalently keep t unchanged, but taking the normal form eases some proofs. We note that $(\!(Q_1Q_2)\!)$ is $(\!(Q_1)\!)\circledcirc(\!(Q_2)\!)$.

We now establish a number of properties that will be used to show the soundness of the interpretation of shallow types, in Lemma 10.3.10. For instance, Property i shows that the interpretation of a type scheme is closed with respect to the instance relation of System F.

Some results below require the argument σ to be in normal form. This requirement is introduced in order to simplify the corresponding proof. Actually, all of these properties are extended to type schemes not in normal forms, once the necessary results have been shown.

Properties 10.3.6

i) *We have* $\mathsf{fsub}\,((\!(\sigma)\!)) = (\!(\sigma)\!)$.

ii) *If* σ *is in normal form and* $\alpha \notin \mathsf{ftv}(\sigma)$*, then* $\forall\,\alpha \cdot (\!(\sigma)\!) \subseteq (\!(\sigma)\!)$.

iii) *If* σ *is in normal form,* $(\!(\sigma)\!)$ *is* $\mathsf{ftv}(\sigma)$*-stable.*

iv) *If* t *is not equivalent to* α*, then* $\mathsf{fsub}\,(t[t'/\alpha]) =_{\alpha} (\mathsf{fsub}\,(t))[t'/\alpha]$.

v) *If* θ *is a substitution and the normal form of* t *is not in* $\mathsf{dom}(\theta)$*, then we have* $\theta(\mathsf{fsub}\,(t)) =_{\mathsf{dom}(\theta)} \mathsf{fsub}\,(\theta(t))$.

vi) *For all* t *in* $(\!(\sigma)\!)$*, we have* $\sigma \leqslant_{/} t$.

vii) *If* $\mathsf{nf}(\sigma) \neq \alpha$ *and* $\mathsf{nf}(\sigma) \neq \bot$*, we have* $\mathsf{fsub}\,((\!(\sigma)\!)[t/\alpha]) =_{\alpha} (\!(\sigma)\!)[t/\alpha]$.

viii) *If* σ *is an F-type in normal form, then* $(\!(\sigma)\!) = \mathsf{fsub}\,(\underline{\sigma})$.

ix) *If the sets* S *and* S' *are* $\bar{\beta}$*-stable, then we have* $\bigcup_{t \in S}^{\bar{\alpha}\#\bar{\beta}} \forall\,\bar{\alpha}\cdot S'[t/\alpha] = \bigcup_{t \in S}^{\bar{\alpha}\#\bar{\beta}\cup\bar{\gamma}} \forall\,\bar{\alpha}\cdot S'[t/\alpha]$

x) *If* θ *is a monotype substitution, then we have* $\theta(\mathsf{fsub}\,(S)) =_{\mathsf{dom}(\theta)} \mathsf{fsub}\,(\theta(S))$.

xi) *If* θ *is a monotype substitution and* σ *is in normal form, then* $\theta((\!(\sigma)\!)) =_{\mathsf{dom}(\theta)} (\!(\theta(\sigma))\!)$ *holds.*

xii) *If* $\sigma_1 \equiv \sigma_2$ *holds, then we have* $(\!(\sigma_1)\!) = (\!(\sigma_2)\!)$.

xiii) *If* θ *is in* $(\!(Q)\!)$*, there exists a System F substitution* θ' *such that* $\theta = \theta' \circ \widehat{Q}(t)$.

See proof in the Appendix (page 299).

Some of the properties above require their argument to be in normal form. Thanks to Property xii, this requirement can be discarded:

Properties 10.3.7

i) *If* $\alpha \notin \mathsf{ftv}(\sigma)$*, then* $\forall\,\alpha \cdot (\!(\sigma)\!) \subseteq (\!(\sigma)\!)$.

ii) *The set* $(\!(\sigma)\!)$ *is* $\mathsf{ftv}(\sigma)$*-stable.*

iii) *If* σ *is an F-type, then* $(\!(\sigma)\!) = \mathsf{fsub}\,(\underline{\sigma})$.

iv) *If* θ *is a monotype substitution, we have* $\theta((\!(\sigma)\!)) =_{\mathsf{dom}(\theta)} (\!(\theta(\sigma))\!)$.

Abstraction is a relation that hides information in the prefix. Its converse relation is revelation, which exposes a type scheme bound to a variable in the prefix, and is possible only by explicit type annotations. These two relations are at the heart of type inference. However, the interpretation of shallow types is not designed for preserving type inference, but only for comparing expressiveness of Shallow ML$^{\mathrm{F}}$ and System F.

Thus abstraction and revelation are ignored by the interpretation, as stated by the following lemma:

Lemma 10.3.8 *If we have a shallow derivation of* $(Q)\ \sigma_1 \mathrel{-\!\!\!\!\sqsubseteq} \sigma_2$*, then for all* θ *in* $(\!(Q)\!)$*, the sets* $\mathsf{fsub}\,(\theta((\!(\sigma_1)\!)))$ *and* $\mathsf{fsub}\,(\theta((\!(\sigma_2)\!)))$ *are equal.*

See proof in the Appendix (page 310).

Note, however, that a corollary of this lemma is that the interpretation of shallow types captures the equivalence relation (as well as the abstraction relation).

The next lemma shows that the interpretation of shallow types is consistent with the instance relation: if σ_2 is an instance of σ_1 in Shallow ML$^{\mathrm{F}}$, then each type in $(\!(\sigma_2)\!)$ must be an instance (in System F) of a type in $(\!(\sigma_1)\!)$. Actually it amounts to having $(\!(\sigma_2)\!) \subseteq (\!(\sigma_1)\!)$. This holds for the instance relation under an empty prefix. The following lemma takes the prefix into account thanks to a substitution θ.

Lemma 10.3.9 *If we have a shallow derivation of* $(Q)\ \sigma_1 \diamondsuit \sigma_2$*, then for all* θ *in* $(\!(Q)\!)$*, we have* $\theta((\!(\sigma_2)\!)) \subseteq \theta((\!(\sigma_1)\!))$*.*

See proof in the Appendix (page 313).

10.3.4 Encoding Shallow F into System F

We consider the Curry-style version of System F, which we call *implicit* System F. Expressions are expressions of the pure lambda-calculus, so that type applications, type abstractions, and type annotations are all implicit. The typing rules of implicit System F are the following:

$$\frac{x : t \in A}{A \vdash^F x : t}\ \text{IF-VAR} \qquad \frac{A, x : t \vdash^F a : t'}{A \vdash^F \lambda(x)\ a : t \to t'}\ \text{IF-FUN} \qquad \frac{A \vdash^F a_1 : t_2 \to t_1 \qquad A \vdash^F a_2 : t_2}{A \vdash^F a_1\ a_2 : t_1}\ \text{IF-APP}$$

$$\frac{A \vdash^F a : t \qquad t \sqsubseteq_F t'}{A \vdash^F a : t'}\ \text{IF-INST} \qquad \frac{A \vdash^F a : t \qquad \alpha \notin \mathsf{ftv}(A)}{A \vdash^F a : \forall\,\alpha \cdot t}\ \text{IF-GEN}$$

As expected, these rules correspond to the rules of System F, where type annotations are removed on lambda-bound variables, type abstractions are type applications are made implicit in rules IF-GEN and IF-INST, respectively.

In order to prove that Shallow F and System F have the same expressiveness, we introduce Shallow UML$^{\mathrm{F}}$ as Shallow F with implicit oracles and without type annotations. The type erasure of an expression typable in Shallow F is typable in Shallow UML$^{\mathrm{F}}$.

Conversely, if an expression is typable in Shallow UML$^{\mathrm{F}}$, it can be annotated in order to be typable in Shallow F. We simply say that Shallow F and Shallow UML$^{\mathrm{F}}$ have the same set of typable terms. We show in the following lemma that a term typable in Shallow UML$^{\mathrm{F}}$ is also typable in implicit System F.

Lemma 10.3.10 *Assume Γ is a type environment with monotypes only. If we have a derivation of $(Q)\ \Gamma \vdash a : \sigma$ in Shallow UML$^{\mathrm{F}}$, then $\theta(\Gamma) \vdash^F a : \theta(t)$ holds for all System F substitutions θ, and System F type t such that $\theta \in (\!(Q)\!)$, and $t \in (\!(\sigma)\!)$.*

Proof: By induction on the shallow derivation of $(Q)\ \Gamma \vdash a : \sigma$. Let θ be in $(\!(Q)\!)$ and t be in $(\!(\sigma)\!)$ $\mathbf{(\underline{1})}$.

∘ Case Var: By hypothesis, we have $\Gamma(x) = \tau$. Hence, $\theta(\Gamma)(x)$ is $\theta(\tau)$. Thus, we can derive $\theta(\Gamma) \vdash^F a : \theta(\tau)$ $\mathbf{(\underline{2})}$ by IF-Var. Additionally, t is in $(\!(\tau)\!)$. Hence, t is in $\mathsf{fsub}\,(\tau)$, which means that t is equivalent to τ. Then $\theta(t)$ is equivalent to $\theta(\tau)$. Thus, $\theta(\Gamma) \vdash^F a : \theta(t)$ holds from (2) by IF-Inst.

∘ Case App: The premises are $(Q)\ \Gamma \vdash a_1 : \tau_2 \to \tau_1$, $(Q)\ \Gamma \vdash a_2 : \tau_2$, a is $a_1\ a_2$, and σ is τ_1. By induction hypothesis, we have $\theta(\Gamma) \vdash^F a_1 : \theta(\tau_2 \to \tau_1)$, as well as $\theta(\Gamma) \vdash^F a_2 : \theta(\tau_2)$. By IF-App, we get $\theta(\Gamma) \vdash^F a : \theta(\tau_1)$ $\mathbf{(\underline{3})}$. We note that $(\!(\tau_1)\!) = \mathsf{fsub}\,(\tau_1)$, hence, t is equivalent to τ_1 (from (1)). This implies $\theta(t)$ equivalent to $\theta(\tau_1)$. Then by IF-Inst and (3), we get $\theta(\Gamma) \vdash^F a : \theta(t)$.

∘ Case Fun: The premise is $(Q)\ \Gamma, x : \tau_0 \vdash a' : \tau$, a is $\lambda(x)\ a'$, and σ is $\tau_0 \to \tau$. We note that $(\!(\tau_0 \to \tau)\!) = \mathsf{fsub}\,(\tau_0 \to \tau)$, hence, t is equivalent to $\tau_0 \to \tau$ (from (1)). Then $\theta(t)$ is equivalent to $\theta(\tau_0 \to \tau)$. By induction hypothesis, we have $\theta(\Gamma, x : \tau_0) \vdash^F a' : \theta(\tau)$. By IF-Fun, we get $\theta(\Gamma) \vdash^F \lambda(x)\ a' : \theta(\tau_0) \to \theta(\tau)$. By IF-Inst, we get $\theta(\Gamma) \vdash^F a : \theta(t)$.

∘ Case Gen: The premise is $(Q, \alpha \diamond \sigma_1)\ \Gamma \vdash a : \sigma_2$ with $\alpha \notin \mathsf{ftv}(\Gamma)$, and σ is $\forall\,(\alpha \diamond \sigma_1)\ \sigma_2$. We consider two subcases:

Subcase σ is $\forall\,(\alpha = \sigma_1)\ \sigma_2$: By definition, we have $(\!(\sigma)\!) = \mathsf{fsub}\big((\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]\big)$. From (1), there exists t_2 in $(\!(\sigma_2)\!)$ such that $t_2[\underline{\sigma_1}/\alpha] \sqsubseteq_F t$ holds. This implies the inclusion $\theta(t_2[\underline{\sigma_1}/\alpha]) \sqsubseteq_F \theta(t)$ $\mathbf{(\underline{4})}$. Let θ' be $\theta \circ [\underline{\sigma_1}/\alpha]$. We note that $\theta' \in (\!(Q, \alpha = \sigma_1)\!)$. By induction hypothesis, we have $\theta'(\Gamma) \vdash^F a : \theta'(t_2)$, that is, $\theta(\Gamma) \vdash^F a : \theta(t_2[\underline{\sigma_1}/\alpha])$. By (4) and IF-Inst, we get $\theta(\Gamma) \vdash^F a : \theta(t)$.

Subcase σ is $\forall\,(\alpha \geq \sigma_1)\ \sigma_2$: By definition, we have

$$(\!(\sigma)\!) = \mathsf{fsub}\left(\bigcup_{t_1 \in (\!(\sigma_1)\!)}^{\bar\alpha \# \mathsf{ftv}(\sigma_1, \sigma_2)} \forall\,\bar\alpha \cdot (\!(\sigma_2)\!)[t_1/\alpha]\right)$$

Let J be $\mathsf{ftv}(\sigma_1, \sigma_2) \cup \mathsf{ftv}(\theta(\Gamma)) \cup \mathsf{dom}(\theta) \cup \mathsf{codom}(\theta)$ $\mathbf{(\underline{5})}$. By Properties 10.3.7.ii and 10.3.6.ix, we get

$$(\!(\sigma)\!) = \mathsf{fsub}\left(\bigcup_{t_1 \in (\!(\sigma_1)\!)}^{\bar\alpha \# J} \forall\,\bar\alpha \cdot (\!(\sigma_2)\!)[t_1/\alpha]\right)$$

From (1), there exist $t_2 \in \llparenthesis\sigma_2\rrparenthesis$, $t_1 \in \llparenthesis\sigma_1\rrparenthesis$, and $\bar\alpha$ disjoint from J such that $\forall\,\bar\alpha\cdot t_2[t_1/\alpha] \sqsubseteq_F t$ $(\underline{\mathbf{6}})$ holds. Let θ' be $\theta\circ[\mathsf{nf}(t_1)/\alpha]$. We note that $\theta' \in \llparenthesis Q, \alpha\ge\sigma_1\rrparenthesis$. By induction hypothesis, we have $\theta'(\Gamma) \vdash^F a : \theta'(t_2)$, that is, $\theta(\Gamma) \vdash^F a : \theta(t_2[\mathsf{nf}(t_1)/\alpha])$. By (5) and IF-Gen, we get $\theta(\Gamma) \vdash^F a : \forall\,\bar\alpha\cdot\theta(t_2[\mathsf{nf}(t_1)/\alpha])$. By (5), this is equal to $\theta(\Gamma) \vdash^F a : \theta(\forall\,\bar\alpha\cdot t_2[\mathsf{nf}(t_1)/\alpha])$. By (6) and F-Inst, we get $\theta(\Gamma) \vdash^F a : \theta(t)$.

∘ Case Inst: The premise is $(Q)\ \Gamma \vdash a : \sigma'$ and $(Q)\ \sigma' \sqsubseteq \sigma$ holds. By Lemma 10.3.9, we have $\theta(\llparenthesis\sigma\rrparenthesis) \subseteq \theta(\llparenthesis\sigma'\rrparenthesis)$. Hence, we have $\theta(t) \in \theta(\llparenthesis\sigma'\rrparenthesis)$, that is, there exists t' in $\llparenthesis\sigma'\rrparenthesis$ such that $\theta(t) = \theta(t')$. By induction hypothesis, we have a derivation of $\theta(\Gamma) \vdash^F a : \theta(t')$, that is, $\theta(\Gamma) \vdash^F a : \theta(t)$.

∘ Case Oracle: The premises are $(Q)\ \Gamma \vdash a : \sigma'$ and $(Q)\ \sigma \sqsubseteq\!\!\!\!- \ \sigma'$. By Lemma 10.3.8, we have $\mathsf{fsub}\,(\theta(\llparenthesis\sigma\rrparenthesis)) = \mathsf{fsub}\,(\theta(\llparenthesis\sigma'\rrparenthesis))$. We note that $\theta(t)$ is in $\mathsf{fsub}\,(\theta(\llparenthesis\sigma\rrparenthesis))$. Hence, $\theta(t)$ is in $\mathsf{fsub}\,(\theta(\llparenthesis\sigma'\rrparenthesis))$, that is, there exists t' in $\llparenthesis\sigma'\rrparenthesis$ such that $\theta(t') \sqsubseteq_F \theta(t)$ $(\underline{\mathbf{7}})$. By induction hypothesis we have $\theta(\Gamma) \vdash^F a : \theta(t')$. By (7) and IF-Inst, we get $\theta(\Gamma) \vdash^F a : \theta(t)$. $\blacksquare$

Note that the above lemma makes sense since $\llparenthesis Q\rrparenthesis$ and $\llparenthesis\sigma\rrparenthesis$ are not empty by construction.

In order to illustrate this result, we consider the expression a equal to `choose id`. In ML^{F}, a has the principal type $\forall\,(\alpha\ge\sigma_{\mathtt{id}})\ \alpha\to\alpha$, which we write σ. The corresponding derivation is the following: (for the sake of readability we only show a meaningful excerpt of the derivation)

$$\textsc{Gen}\;\dfrac{\textsc{App}\;\dfrac{\textsc{Inst}\;\dfrac{\begin{array}{c}(\alpha\ge\sigma_{\mathtt{id}})\ \emptyset\vdash \mathtt{choose} : \forall\,(\beta)\ \beta\to\beta\to\beta\\ (\alpha\ge\sigma_{\mathtt{id}})\ \forall\,(\beta)\ \beta\to\beta\to\beta \sqsubseteq \alpha\to\alpha\to\alpha\end{array}}{(\alpha\ge\sigma_{\mathtt{id}})\ \emptyset\vdash \mathtt{choose} : \alpha\to\alpha\to\alpha} \qquad \dfrac{\begin{array}{c}(\alpha\ge\sigma_{\mathtt{id}})\ \emptyset\vdash \mathtt{id} : \sigma_{\mathtt{id}}\\ (\alpha\ge\sigma_{\mathtt{id}})\ \sigma_{\mathtt{id}} \sqsubseteq \alpha\end{array}}{(\alpha\ge\sigma_{\mathtt{id}})\ \emptyset\vdash \mathtt{id} : \alpha}\;\textsc{Inst}}{(\alpha\ge\sigma_{\mathtt{id}})\ \emptyset\vdash a : \alpha\to\alpha}}{(\emptyset)\ \emptyset\vdash a : \forall\,(\alpha\ge\sigma_{\mathtt{id}})\ \alpha\to\alpha}$$

The above lemma states that for any System F type t in $\llparenthesis\sigma\rrparenthesis$, $\vdash^F a : t$ holds (the prefix is empty). We write $\mathtt{id}_\alpha$ for $\alpha\to\alpha$, and $t_{\mathtt{id}}$ for $\forall\,\alpha\cdot\mathtt{id}_\alpha$. We give two derivations corresponding to two choices of t, namely $t_{\mathtt{id}}\to t_{\mathtt{id}}$ and $\forall\,\alpha\cdot\mathtt{id}_\alpha\to\mathtt{id}_\alpha$.

$$\text{IF-App}\;\dfrac{\text{IF-Inst}\;\dfrac{\begin{array}{c}\vdash^F \mathtt{choose} : \forall\,\beta\cdot\beta\to\beta\to\beta\\ \forall\,\beta\cdot\beta\to\beta\to\beta \sqsubseteq_F t_{\mathtt{id}}\to t_{\mathtt{id}}\to t_{\mathtt{id}}\end{array}}{\vdash^F \mathtt{choose} : t_{\mathtt{id}}\to t_{\mathtt{id}}\to t_{\mathtt{id}}} \qquad \vdash^F \mathtt{id} : t_{\mathtt{id}}}{\vdash^F a : t_{\mathtt{id}}\to t_{\mathtt{id}}}$$

$$\text{IF-INST}\;\frac{\begin{array}{c}\vdash^{F} \mathtt{choose} : \forall\,\beta\cdot\beta\to\beta\to\beta \\ \forall\,\beta\cdot\beta\to\beta\to\beta \sqsubseteq_{F} \mathtt{id}_{\alpha}\to\mathtt{id}_{\alpha}\to\mathtt{id}_{\alpha}\end{array}}{\vdash^{F} \mathtt{choose} : \mathtt{id}_{\alpha}\to\mathtt{id}_{\alpha}\to\mathtt{id}_{\alpha} \quad (\underline{\mathbf{8}})}$$

$$\begin{array}{l}\text{IF-INST}\\ \dfrac{\vdash^{F} \mathtt{id} : t_{\mathtt{id}} \qquad t_{\mathtt{id}} \sqsubseteq_{F} \mathtt{id}_{\alpha}}{\vdash^{F} \mathtt{id} : \mathtt{id}_{\alpha} \quad (\underline{\mathbf{9}})}\end{array}$$

$$\text{IF-GEN}\;\frac{\text{IF-APP}\;\dfrac{(8) \qquad (9)}{\vdash^{F} a : \mathtt{id}_{\alpha}\to\mathtt{id}_{\alpha}}}{\vdash^{F} a : \forall\,\alpha\cdot\mathtt{id}_{\alpha}\to\mathtt{id}_{\alpha}}$$

In the first derivation, the substitution taken from $(\!(\alpha \geq \sigma_{\mathtt{id}})\!)$ is $[t_{\mathtt{id}}/\alpha]$, while it is $[\mathtt{id}_{\alpha}/\alpha]$ in the second derivation.

As a conclusion, each term typable in Shallow $\mathrm{UML}^{\mathrm{F}}$ is also typable in System F, thus System F and Shallow F have the same set of typable terms. As a corollary, type inference in Shallow $\mathrm{UML}^{\mathrm{F}}$ is undecidable, and so is type inference in Shallow $\mathrm{ML}^{\mathrm{F}}_{\star}$. However, this does not imply undecidability of type inference in $\mathrm{ML}^{\mathrm{F}}_{\star}$, even though the converse would be surprising.

Since Shallow F and System-F have the same expressiveness, for every term a of Shallow F there exists a term M of explicitly typed system F that has the same erasure as a. However, there is no unicity of a.

The interpretation of shallow types in System F suggests a semantics for types. However, the encoding $(\!(\sigma)\!)$ of shallow types is disappointing: On the one hand, as stated by Lemma 10.3.8, abstraction and revelation are not covered by the interpretation. On the other hand, the strength of ML^{F} lies in the way it propagates type information via abstraction and revelation, which makes type inference possible. Thus, the given encoding does not provide a semantics meaningful for type inference. Fixing this amounts to finding an interpretation that makes the difference between *e.g.* $\forall\,(\alpha = \sigma_{\mathtt{id}})\ \alpha \to \alpha$ and $\forall\,(\alpha = \sigma_{\mathtt{id}}, \beta = \sigma_{\mathtt{id}})\ \alpha \to \beta$. We are not aware of any such semantics.

10.4 Discussion

Let us summarize the results shown above. This document studies ML^{F} (with type annotations) and $\mathrm{ML}^{\mathrm{F}}_{\star}$ (with oracles), which have the same set of typable terms. We wish to compare ML^{F} and System F. We have already shown in Chapter 9 that System F can be encoded in ML^{F}. Hence, the set of typable terms of System F is included in the set of typable terms of ML$^{\mathrm{F}}$. We simply say that System F is included in ML$^{\mathrm{F}}$. We do not show the converse inclusion, but rather consider a restriction of ML$^{\mathrm{F}}$, called Shallow ML$^{\mathrm{F}}$. In Section 10.2, we have shown that a term is typable in Shallow ML$^{\mathrm{F}}$ if and only if it is shallow and typable in ML$^{\mathrm{F}}$. Thus, Shallow ML$^{\mathrm{F}}$ can as well be defined as ML$^{\mathrm{F}}$ where type annotations are required to be types of System F only. As a consequence, System F is included in Shallow ML$^{\mathrm{F}}$. More precisely, terms of System F do not have the `let` construct, thus System F is included in Shallow F, which is Shallow ML$^{\mathrm{F}}$ without `let`. Conversely, we have shown in Section 10.3, that Shallow F is included in System F. Hence, Shallow F and System F have the same set of typable terms. This can be pictured by the following sequence of relations:

$$\text{ML}^{\text{F}} = \text{ML}^{\text{F}}_{\star} \supseteq \text{Shallow ML}^{\text{F}} \supseteq \text{Shallow F} = \text{System F}$$

A remaining question is whether Shallow F is a strict subset of Shallow ML$^{\mathrm{F}}$ (encoding `let` constructs as λ-abstractions). We conjecture that this is true.

More precisely, our candidate for discrimination is the term a_0 given below. We first introduce some notations that make the term a_0 more readable. Given a number n, we write $\bar{n}$ for the church numeral n, that is, $\lambda(f)\ \lambda(x)\ (f\ \ldots (f\ x))$ with f repeated n times. We write S the successor function, that is, $\lambda(n)\ \lambda(f)\ \lambda(x)\ f\ (n\ f\ x)$. We recall that `auto` is $\lambda(x)\ x\ x$. Additionally, given two terms a and a', the sequence $a; a'$ is syntactic sugar for $\bar{0}\ a\ a'$. Then the term a_0 is defined as $\texttt{let}\ x = a_1\ \texttt{in}\ a_2$ where a_1 is the function $\lambda(f)\ f\ \texttt{id}$ and a_2 is the sequence $x\ \texttt{auto}\ ;\ (\lambda(y)\ x\ y\ ;\ y\ S)\ (\lambda(z)\ \lambda(z')\ z\ \bar{2})$.

The term a_0 is in Shallow ML$^{\mathrm{F}}$. However, we conjecture that a_0 is not typable in System F. The intuition is that the two occurrences of x have "incompatible types" $((\forall\,\alpha \cdot (\alpha \to \alpha)) \to t) \to t$ for some type t and $(\mathbb{N}^2 \to \mathbb{N}^2) \to \mathbb{N}^2$ in System F where $\mathbb{N}$ is the type $\forall\,\alpha \cdot (\alpha \to \alpha) \to \alpha \to \alpha$ of church numerals and $\mathbb{N}^2$ is $\mathbb{N} \to \mathbb{N}$. In Shallow ML$^{\mathrm{F}}$, x can be assigned the type $(\forall\,(\beta, \alpha \geq \sigma_{\texttt{id}})\ \alpha \to \beta) \to \beta$.

Furthermore, assuming that a is not in System F, we can build the term a' equal to $\bar{1}\ (\lambda(x)\ a_2)\ a_1$, which is in ML$^{\mathrm{F}}$ but not in Shallow ML$^{\mathrm{F}}$. Indeed, $(\lambda(x)\ a_2)\ a_1$ is typable in ML$^{\mathrm{F}}$ (annotating x with the type given above), thus a' is typable, as shown in Example 5.2.11. However, a' is not typable in Shallow ML$^{\mathrm{F}}$. Indeed, it would otherwise be typable in Shallow F (it does not use `let`), and therefore it would be typable in System F. This has been conjectured not to hold.

Still, ML$^{\mathrm{F}}$ remains a second-order system and in that sense should not be *significantly* more expressive than System F. In particular, we conjecture that the term $(\lambda(y)\ y\ I\ ;\ y\ K)\ (\lambda(x)\ x\ x)$ that is typable in F$^{\omega}$ but not in F [GR88] is not typable in ML$^{\mathrm{F}}$ either. Conversely, we do not know whether there exists a term of ML$^{\mathrm{F}}$ that is not typable in F$^{\omega}$.

Notice however that the term a_0 given above is typable in F^{ω}. The hint is to encode the type $\forall\,(\beta)\;\forall\,(\alpha'{\geq}\forall\,(\alpha)\;\alpha \to \alpha)\;(\alpha' \to \beta) \to \beta$ as $\forall\,A{\cdot}\forall\,\beta{\cdot}((\forall\,\alpha{\cdot}A\;\alpha \to A\;\alpha) \to \beta) \to \beta$ and use $\Lambda(\gamma)\;\gamma$ or $\Lambda(\gamma)\;\mathbb{N}$ for A to recover the types of the two occurrences of x. Then a can be typed as follows in F^{ω}: $(\lambda(x : \forall\,A \cdot \forall\,\beta \cdot ((\forall\,\alpha \cdot A\;\alpha \to A\;\alpha) \to \beta) \to \beta)\;a'_2)\;a'_1$, where, a'_1 is $\Lambda(A)\;\Lambda(\beta)\;\lambda(f : (\forall\,\alpha \cdot A\;\alpha \to A\;\alpha) \to \beta)\;f\;(\Lambda(\alpha)\;\mathtt{id}[A\;\alpha])$ and a'_2 is

$$\begin{array}{l}
\Lambda(\alpha) \\
\quad x\;[\Lambda(\beta)\;\beta]\;[\mathtt{id}_{\alpha}]\;(\lambda(x : \sigma_{\mathtt{id}})\;x\;[\mathtt{id}_{\alpha}]\;x[\alpha])\;; \\
\quad (\lambda(y : \mathbb{N}^2 \to \mathbb{N}^2)\;x\;[\Lambda(\alpha)\;\mathbb{N}^2]\;[\mathbb{N}]\;y\;;\;y\;S) \\
\qquad\quad (\lambda(z : \mathbb{N}^2)\;\lambda(z' : \mathbb{N})\;z\;\bar{2})
\end{array}$$

This suggests that there may exist an encoding of ML^{F} into F^{ω}.

We can now enrich the above diagram with these conjectures:

$$\begin{tikzcd}
 & & \mathrm{F}^{\omega} & & \\
\begin{pmatrix} \mathrm{ML}^{\mathrm{F}} \\ \mathrm{ML}^{\mathrm{F}}_{\star} \end{pmatrix} \arrow[urr, "\subset^{(\underline{\mathbf{12}})}"] & \supset^{(\underline{\mathbf{10}})} & \begin{pmatrix} \text{Shallow } \mathrm{ML}^{\mathrm{F}} \\ \text{Shallow } \mathrm{ML}^{\mathrm{F}}_{\star} \end{pmatrix} & \supset^{(\underline{\mathbf{11}})} & \text{Shallow F} = \text{System F} \arrow[ull, "\supset^{(\underline{\mathbf{13}})}"']
\end{tikzcd}$$

The inclusion (11) corresponds to the encoding of System F, which is shallow (proved). It is conjectured to be a strict inclusion. The inclusion (10) is by construction. It is strict whenever (11) is strict. The inclusion (13) is by construction of F^{ω}. It is known to be a strict inclusion [GR88]. We expect the inclusion (12) to hold, as suggested by the example above, however we are not able to give a systematic encoding of ML^{F} into F^{ω}. This inclusion is also conjectured to be strict.

Reducing all let-bindings in a term of Shallow ML^{F} produces a term in Shallow F. Hence, terms of Shallow ML^{F} are strongly normalizable. We conjecture that so are all terms of ML^{F}.

Chapter 11

Language extensions

We have defined ML^{F} as a core language made of the expressions of ML and type annotations. We have shown that all programs in ML are still typable in ML^{F}, without any annotation. However, real ML programs often mention useful language extensions. We show in this chapter how such extensions could be added to ML^{F}.

Some constructs, such as tuples or records, are looked at in Section 11.1 and are easily added to ML^{F}. Interestingly, first-class polymorphism allows for a direct encoding of some constructs (such as pairs), which is not always possible in ML. Imperative features are considered in Section 11.2: References are known to misbehave in the presence of ML-style polymorphism, thus this extension is more delicate and requires some restrictions. More precisely, we consider the well known value restriction and a more subtle variant, the *relaxed value restriction.* In Section 11.3, we consider a useful feature that is not meaningful in ML because type annotations are not required for type inference. This feature consists of mechanically propagating type annotations from top-level to subexpressions. As an immediate application, it can be used to automatically propagate type annotations from interface files to implementation files. For instance, the type information given in the signature of a module can be automatically propagated to each element and each subexpression of the module.

11.1 Tuples, Records

Because the language is parameterized by constants, which can be used either as constructors or primitive operations, the language can import foreign functions defined via appropriate δ-rules. These could include primitive types (such as integers, strings, *etc.*) and operations over them. Sums and products, as well as predefined datatypes, can also be treated in this manner, but some (easy) extension is required to declare new data-types within the language itself.

As an example, the primitives that handle pairs have the same types as in ML:

$$\begin{array}{rcl} (\_,\_) & : & \forall\,(\alpha,\beta)\ \alpha \to \beta \to \alpha \times \beta \\ \texttt{fst} & : & \forall\,(\alpha,\beta)\ \alpha \times \beta \to \alpha \\ \texttt{snd} & : & \forall\,(\alpha,\beta)\ \alpha \times \beta \to \beta \end{array}$$

The corresponding δ-rules are

$$\begin{array}{lcl} \texttt{fst}\ (v_1, v_2) & \longrightarrow & v_1 \\ \texttt{snd}\ (v_1, v_2) & \longrightarrow & v_2 \end{array}$$

We could also introduce pairs using the usual encoding of pairs in System F:

$$\begin{array}{lcl} (\_,\_) & = & \lambda(x)\ \lambda(y)\ \lambda(f)\ f\ x\ y \\ \texttt{fst} & = & \lambda(p)\ p\ (\lambda(x)\ \lambda(y)\ x) \\ \texttt{snd} & = & \lambda(p)\ p\ (\lambda(x)\ \lambda(y)\ y) \end{array}$$

However, as remarked by Fritz Henglein [Hen93], this encoding is not correct in ML. Indeed, the type given to $\lambda(p)\ (\texttt{fst}\ p, \texttt{snd}\ p)$ is then $\forall\,\alpha \cdot \alpha \times \alpha \to \alpha \times \alpha$, whereas it is $\forall\,\alpha\beta \cdot \alpha \times \beta \to \alpha \times \beta$ when using the primitive pairs. This cannot be fixed in ML, but it can in ML$^{\mathsf{F}}$, adding a type annotation to the argument of `fst` and `snd`:

$$\begin{array}{lcl} \texttt{fst} & = & \lambda(p : \exists\,(\alpha,\beta)\ \forall\,(\gamma)\ (\alpha \to \beta \to \gamma) \to \gamma)\ p\ (\lambda(x)\ \lambda(y)\ x) \\ \texttt{snd} & = & \lambda(p : \exists\,(\alpha,\beta)\ \forall\,(\gamma)\ (\alpha \to \beta \to \gamma) \to \gamma)\ p\ (\lambda(x)\ \lambda(y)\ y) \end{array}$$

Then the pairs encoded as functions and the primitive pairs have the same typing behavior.

11.2 References

To give an account of references, the simplest modification of the dynamic semantics is to use a global store (mapping store locations to values) and a small-step reduction relation over store-expression pairs. This carries over to ML$^{\mathsf{F}}$ without any difficulty.

Adapting the static semantics to ensure that store locations are always used at the same type is more difficult. In ML, this can be done by restricting implicit let-polymorphism to syntactic values [Wri95]. Hence, the type of a location, which must be the type of the non-value expression creating the location, cannot be polymorphic. This solution can be adapted to ML$^{\mathsf{F}}$ as well.

Value restriction The value restriction consists of restricting Rule Gen to values (non-expansive expressions) only:

Gen-vr

$$\frac{(Q, \alpha \diamond \sigma)\ \Gamma \vdash a : \sigma' \qquad \alpha \notin \mathsf{ftv}(\Gamma) \qquad a \text{ non-expansive}}{(Q)\ \Gamma \vdash a : \forall\,(\alpha \diamond \sigma)\ \sigma'}$$

Then the type inference algorithm (page 164) is modified as follows:

Case $a\,b$:

- **let** $(Q_1, \sigma_a) = \texttt{infer}\,(Q, \Gamma, a)$
- **let** $(Q_2, \sigma_b) = \texttt{infer}\,(Q_1, \Gamma, b)$
- **let** $\alpha_a, \alpha_b, \beta \notin \mathsf{dom}(Q_2)$
- **let** $Q_3 = \texttt{unify}\,((Q_2, \alpha_a \geq \sigma_a, \alpha_b \geq \sigma_b, \beta \geq \bot), \alpha_a, \alpha_b \to \beta)$ **return** (Q_3, β)

We return Q_3 whereas, in the previous version of the algorithm, Q_3 is split in two parts Q_4 and Q_5. Then Q_4 is returned as the new prefix, and Q_5 is generalized in front of β.

However, this solution is likely to be very disappointing when applied to ML^{F}, whose machinery relies on type generalization and the use of polymorphic non-functional expressions (which cannot be η-expanded). As a first example, type annotations are primitives that reveal polymorphism upon application. If the value-restriction holds, a type annotation primitive does no longer reveal polymorphism, because generalization is forbidden. Hence, the introduction of the value-restriction requires a dedicated typing rule for type annotations. Fortunately, there is a simple relaxation of value-only polymorphism that allows type variables to remain polymorphic in the type of non-values, as long as they only occur covariantly [Gar02]. Since this extension was designed for Poly-ML in the first place (even though it is more general), we expect this solution to work well for ML^{F} as well and to be sufficient in practice.

Relaxed value restriction Given a flat type t, we define $\mathsf{ftv}^-(t)$ inductively as follows:

$$\mathsf{ftv}^-(\alpha) = \emptyset \qquad \mathsf{ftv}^-(\bot) = \emptyset \qquad \mathsf{ftv}^-(t_1 \to t_2) = \mathsf{ftv}(t_1) \cup \mathsf{ftv}^-(t_2)$$

Given a type scheme σ, $\mathsf{ftv}^-(\sigma)$ is by definition $\mathsf{ftv}^-(\mathsf{proj}(t))$. We note that all free variables of σ occurring at the left of an arrow are in $\mathsf{ftv}^-(\sigma)$. This definition of negative occurrences is not usual, since α is considered negative in $(\alpha \to \beta) \to \beta$, while it is often considered positive in other type systems.

The relaxed value restriction consists of restricting Rule Gen as follows:

$$\textsc{Gen-rvr}\quad \frac{(Q, \alpha \diamond \sigma)\;\Gamma \vdash a : \sigma' \qquad \alpha \notin \mathsf{ftv}(\Gamma) \qquad a \text{ non-expansive or } \alpha \notin \mathsf{ftv}^-(\sigma')}{(Q)\;\Gamma \vdash a : \forall\,(\alpha \diamond \sigma)\;\sigma'}$$

In order to implement this rule in the type inference algorithm, we need to modify the split algorithm (defined page 112) according to the occurrence of variables. Hence the following definition:

Definition 11.2.1 The positive *splitting* of a closed prefix Q according to the set $\bar{\alpha}$ and the type scheme σ is written $Q{\uparrow}^{\sigma}\bar{\alpha}$ and defined inductively as follows (we require $\bar{\alpha} \subseteq \mathsf{dom}(Q)$).:

$$\emptyset{\uparrow}^{\sigma}\bar{\alpha} = (\emptyset, \emptyset)$$

$$(Q, \alpha \diamond \sigma){\uparrow}^{\tau}\bar{\alpha} = \begin{cases} (\alpha \diamond \sigma) \xrightarrow{1} Q{\uparrow}^{\sigma'}(\bar{\alpha} - \alpha) \cup \mathsf{ftv}(\sigma) & \text{when } \alpha \in \bar{\alpha} \text{ or } \alpha \in \mathsf{ftv}^{-}(\sigma') \\ (\alpha \diamond \sigma) \xrightarrow{2} Q{\uparrow}^{\forall\,(\alpha\diamond\sigma)\sigma'}\bar{\alpha} & \text{otherwise} \end{cases}$$

□

Then the type-inference algorithm is modified as follows:

Case $a\ b$:

- **let** $(Q_1, \sigma_a) = \mathtt{infer}\ (Q, \Gamma, a)$
- **let** $(Q_2, \sigma_b) = \mathtt{infer}\ (Q_1, \Gamma, b)$
- **let** $\alpha_a, \alpha_b, \beta \notin \mathsf{dom}(Q_2)$
- **let** $Q_3 = \mathtt{unify}\ ((Q_2, \alpha_a \geq \sigma_a, \alpha_b \geq \sigma_b, \beta \geq \bot), \alpha_a, \alpha_b \to \beta)$
- **let** $(Q_4, Q_5) = Q{\uparrow}^{\beta}\,\mathsf{dom}(Q)$
- **return** $(Q_4, \forall\,(Q_5)\ \beta)$

The only difference is that variables that are generalized (namely Q_5) are those obtained from the positive splitting of Q, which takes the occurrence of the variables into account.

This restriction has been implemented and seems to work well on a few significant examples. It remains to find if it is usable in practice for larger programs.

11.3 Propagating type annotations

In ML$^{\mathrm{F}}$, type annotations are only needed on arguments of λ-abstractions that are used polymorphically. However, for documentation purposes or for readability, it can be useful to put extra type annotations. For instance, consider the following top-level definition: `let delta` $= \lambda(x : \sigma_{\mathtt{id}})\ x\ x$. One could wish to give explicitly the type of `delta` (the annotation $\sigma_{\mathtt{id}} \to \sigma_{\mathtt{id}}$ is syntactic sugar for $\forall\,(\alpha = \sigma_{\mathtt{id}})\ \forall\,(\beta = \sigma_{\mathtt{id}})\ \alpha \to \beta$): `let` (`delta` : $\sigma_{\mathtt{id}} \to \sigma_{\mathtt{id}}$) $= \lambda(x : \sigma_{\mathtt{id}})\ x\ x$ (**<u>14</u>**). In this case, the two annotations are obviously redundant. We would like to get rid of the second one ($x : \sigma_{\mathtt{id}}$), which is subsumed by the first one.

In a first time, we define some syntactic sugar to give meaning to the notation `let` $(x : \sigma) = a$. Then we show how type annotations can be automatically propagated `in` order to avoid useless redundancy.

$\texttt{let}$-bound variables and λ-bound variables can be annotated, using the following syntactic sugar (the first rule was already defined in Section 8.3):

$$\begin{array}{rcl}
\lambda(x:\exists\,(Q)\;\sigma)\;a & \longrightarrow & \lambda(x)\;\texttt{let}\;x=(x:\exists\,(Q)\;\sigma)\;\texttt{in}\;a\\
\texttt{let}\;(x:\exists\,(Q)\;\sigma)=a\;\texttt{in}\;a' & \longrightarrow & \texttt{let}\;x=(a:\exists\,(Q)\;\sigma)\;\texttt{in}\;a'\\
\texttt{let rec}\;(x:\exists\,(Q)\;\sigma)=a\;\texttt{in}\;a' & \longrightarrow & \texttt{let rec}\;\;x=\\
 & & \texttt{let}\;x=(x:\exists\,(Q)\;\sigma)\;\texttt{in}\\
 & & (a:\exists\,(Q)\;\sigma)\;\texttt{in}\;a'
\end{array}$$

Next, we show how the outer type annotation in (14) can be propagated to the inner λ-abstraction. More generally, we show how type annotations are propagated to inner terms. Since flexible bindings and rigid bindings are quite different in nature, they are propagated differently. The flexible domain and rigid domain of a prefix, defined next, clearly separate those two kinds of bindings.

Definition 11.3.1 The flexible domain and the rigid domain of a prefix Q , written $\mathsf{dom}_{>}(Q)$ and $\mathsf{dom}_{=}(Q)$ respectively, are defined as follows:

- $\alpha \in \mathsf{dom}_{>}(Q)$ if and only if $(\alpha \geq \sigma) \in Q$ or $(\alpha = \sigma') \in Q$ and $\sigma' \in \mathcal{T}$.
- $\alpha \in \mathsf{dom}_{=}(Q)$ if and only if $(\alpha = \sigma) \in Q$ or $(\alpha \geq \sigma') \in Q$ and $\sigma' \in \mathcal{T}$. □

We note that $\mathsf{dom}(Q)$ is the union of $\mathsf{dom}_{>}(Q)$ and $\mathsf{dom}_{=}(Q)$, and that $\mathsf{dom}(\widehat{Q})$ is their intersection. Indeed, as explained in Definition 1.5.7, bindings $(\alpha \diamond \sigma)$ where $\sigma \in \mathcal{T}$ are considered both flexible and rigid.

The rules given below syntactically propagate type annotations. To ease readability, we do not keep the original type annotation in place but only show the propagation mechanism. Actually, the outer type annotation is meant to be kept in place *and* propagated.

$$\begin{array}{rcl}
(a_1\;a_2:\exists\,(Q)\;\sigma) & \longrightarrow & (a_1:\exists\,(\alpha,Q)\;\forall\,(\alpha'=\sigma)\;\alpha\to\alpha')\;a_2\\
(\texttt{let}\;x=a_1\;\texttt{in}\;a_2:\exists\,(Q)\;\sigma) & \longrightarrow & \texttt{let}\;x=a_1\;\texttt{in}\;(a_2:\exists\,(Q)\;\sigma)\\
(a_1,a_2:\exists\,(Q)\;\forall\,(Q')\;\tau_1\times\tau_2) & \longrightarrow & ((a_1:\exists\,(Q)\;\forall\,(Q')\;\tau_1),\\
 & & \qquad\quad (a_2:\exists\,(Q)\;\forall\,(Q')\;\tau_2))\\
(\lambda(x)\;a:\exists\,(Q)\;\forall\,(Q')\;\tau_1\to\tau_2) & \longrightarrow & \lambda(x:\exists\,(QQ_1)\;\forall\,(Q'_1)\;\tau_1)\\
 & & \qquad\quad (a:\exists\,(QQ_2)\;\forall\,(Q'_2)\;\tau_2)\\
 & & \text{where}\\
 & & (Q_1,Q'_1)\stackrel{\triangle}{=}Q'\!\uparrow\mathsf{ftv}(\tau_2)\cup\mathsf{dom}_{>}(Q')\\
 & & (Q_2,Q'_2)\stackrel{\triangle}{=}Q'\!\uparrow\mathsf{ftv}(\tau_1)
\end{array}$$

Hence, if the expression $\texttt{let}\;(\texttt{delta}:\sigma_{\texttt{id}}\to\sigma_{\texttt{id}})=\lambda(x)\;x\;x$ is given in input, it is first replaced by $\texttt{let delta}=(\lambda(x)\;x\;x:\forall\,(\alpha=\sigma_{\texttt{id}},\beta=\sigma_{\texttt{id}})\;\alpha\to\beta)$. Then the type annotation is propagated as follows (writing type annotations in normal form): $\texttt{let}\,\texttt{delta} = (\lambda(x : \sigma_{\texttt{id}})\ ((x : \exists\,(\alpha)\ \alpha \to \sigma_{\texttt{id}})\ x : \sigma_{\texttt{id}}) : \forall\,(\alpha = \sigma_{\texttt{id}}, \beta = \sigma_{\texttt{id}})\ \alpha \to \beta)$. The type annotation $(\_ : \sigma_{\texttt{id}})$ has been automatically propagated to the λ-bound variable x. Note that the expression $\texttt{let}\ (\texttt{delta} : \exists\,(\alpha)\ \sigma_{\texttt{id}} \to \alpha) = \lambda(x)\ x\ x$ is also typable by syntactic propagation of type annotations.

Chapter 12

ML^{F} in practice

One of the main motivations for designing ML^{F} was that the amount of explicit type information required in some previous proposals was too big. Satisfyingly, the encoding of System F is light, and we know that arguments of λ-abstractions that are not used polymorphically need not be annotated.

12.1 Some standard encodings

Using the prototype, we were able to test a few simple examples such as the usual encodings of Church integers, booleans, and tuples in System F. As expected, very few type annotations are needed, and always at predictable places. Indeed, for arguments that are known to have a polymorphic type (for example the polymorphic type encoding integers), it is usually good practice to annotate them using an abbreviation: on the one hand, such arguments are most of the time more “significant” than monomorphic arguments; on the other hand, the inferred types will be much easier to read, thanks to the abbreviation. Besides, although some functions abstract over polymorphic types, they do not necessarily use their argument in a polymorphic way. As a result, a substantial number of function definitions does not actually need the type annotation that the programmer already provided for his own convenience.

In the following example, we define Church integers and some operations over them.[1]

```
# type Int = ∀α. (α → α) → (α → α)
type Int defined.

# let succ (n:Int) = fun f x → n f (f x)
val succ : ∀(β ≥ ∀α. (α → α) → α → α) Int → β
```

---

[1] The output is taken from the prototype, but slightly modified to make it more readable.

```
# let add (n:Int) (m:Int) = n succ m
val add : ∀(β ≥ Int → Int) Int → β
```

Actually, both of these functions can be written without type annotations (indeed, they are typable in ML). But then the inferred type, which is equivalent, is harder to read:

```
# let add n m = n succ m
val add : ∀(α ≥ ∀(γ ≥ ∀ x.(x → x) → x → x) Int → γ)
     ∀(δ) ∀(ϵ) (α → δ → ϵ) → δ → ϵ
```

Moreover, it sometimes happens that, although a function is typable without any type annotation, the type annotation is mandatory in order to get the "correct" type (see the example with pairs in Section 11.1). Indeed, the principal type of an expression depends on the polymorphic type annotations.

Notice that in all of these cases, the annotation could be propagated from the signature file to the implementation, using the technique described in Section 11.3.

We admit that these examples are only simple encodings that do not make strong use of first-class polymorphism. Thus, it is not surprising that so few annotations are actually needed. However, it would be possible to build more involved examples by using these encodings with complex data structures. For example, it is possible to store Church integers (which are polymorphic) in a hashtable without any need for type annotations. The following piece of code typechecks in ML^{F} with the value restriction.

```
let table = Hashtbl.create 10
Hashtbl.add table "one" (λ(f) λ(x) f x)
Hashtbl.add table "two"  (λ(f) λ(x) f (f x))
add (Hashtbl.find table "one") (Hashtbl.find table "two")
```

The last line applies `add`, which requires its two arguments to be polymorphic of type `Int` (Church integers). The type of `table` is monomorphic, namely α `hashtable`, where α is a *weak* type variable. Still, α is bound to `Int` in the prefix. We see that the type parameter of the type constructor `hashtable` is implicitly instantiated by (a type variable standing for) a polymorphic type. This makes hashtables, or any data structure, compositionnally usable, including with polymorphic values.

12.2 Existential Types

Another interesting example is the encoding of existentials. In System F, the existential type $\exists\alpha.t$ is isomorphic to the polymorphic type $\forall\,\beta\cdot(\forall\,\alpha\cdot t\to\beta)\to\beta$. Then, an existential value v with type $\exists\alpha.t$ is encoded as a function expecting a polymorphic argument: `let` $v' = \Lambda(\beta)\ \lambda(x:\forall\,\alpha\cdot t\to\beta)\ x\ v$.

In order to open an existential v', one must apply it to a polymorphic function, that is, a function parametric over the type being abstracted: (we omit the type application) v' [..] $(\Lambda(\alpha)\ \lambda(x:\alpha)\ (x,x))$.

In ML$^{\text{F}}$, the encoding of existentials gets a real benefit from type inference. Indeed, while the creation of the existential itself requires a type annotation, the opening does not require any type annotation. This is the best we can expect since the abstract type given to the encapsulated value can only be defined by the programmer himself.

Let us consider a concrete example: we build an encapsulation for *tasks*, that is, a pair of a function and its argument. Tasks can be put in a list and evaluated by a server. The server opens each encapsulation and applies the function (second element of the pair) to its argument (first element of the pair).

```
let encapsulate v = λ(f : ∀α. α × (α → unit) → β) f v
let create f arg = encapsulate (arg, f)

let serve box = box (λp (snd p) (fst p))
```

New tasks are easily created using `create`, put in any data structure such as a list, and served using `serve`:

```
let taskList = (create f1 arg1) :: (create f2 arg2) :: ...
let server () = List.iter serve taskList
```

Notice that only a single type annotation is needed for the whole example.

12.3 When are annotations needed?

ML$^{\text{F}}$ motto is “annotate arguments that are used polymorphically”. This does not mean that the argument is used twice. Indeed, in some examples, an annotation is required on an argument which is only used once.

Consider the following example:

```
let auto (x : ∀α. α → α) = x x
let t z = auto z
t id
```

`auto` is defined as usual: it requires a polymorphic argument. Then, we define `t` that expects a polymorphic argument `z` and applies `auto` to it. The argument `z` is not annotated because it is not used polymorphically, but only passed through.

We consider a variant `auto`$_2$ defined as follows:

```
let auto₂ (z:∀α. unit → α → α) = (z ()) (z ())
 : ∀(β = ∀α. unit → α → α) ∀(γ ≥ ∀α. α → α) β → γ
```

Finally, we build `t`$_2$ that, intuitively, takes the identity as an argument, builds a λ-abstraction, and applies `auto`$_2$ to the result.

```
let t₂ y = auto₂ (λ() y)
```

This expression is not typable in ML^F. It is an example of a function which uses its argument only once, but which requires an annotation. Indeed, annotating `y` with $\forall(\alpha)\ \alpha \to \alpha$ suffices to make the expression typable. In this example, the argument `y` is used polymorphically in the sense that it is used to build a function which is required to be polymorphic. On the contrary, in $\lambda z.$`auto` z, the argument z is not used but only passed through.

12.4 A detailed example

In the following example, we aim at writing some generic functions that can be used with different kinds of data structures. For instance, we define the functions `size`, `sum` and `max` that return, respectively, the number of elements of the given data structure, the sum of all (integers) elements of the data structure, and the maximal element of the structure. These functions are written only once, but are applied on two different incompatible implementations. The first one is a list of integers, the second one is a balanced binary tree. Both of these data structures are viewed abstractly, using their associated iterator.

The following type definition corresponds to the type of iterators over type structures holding integers. Since such data structures are typically special instances of graphs, we simply call this type `graph`.

```
type graph = ∀α. (int → α → α) → α → α
```

The first data structure we use is a list. The iterator over lists is easily defined as follows[2]:

```
let rec list_iter list f accu =
  if list = [] then accu
  else list_iter (cdr list) f (f (car list) accu)
 : ∀α. ∀β. α list → (α → β → β) → β → β
```

The second data structure is a balanced binary tree. We define a datatype for such trees and the corresponding iterator.

---

[2] This full example is written as such in our prototype implementation of ML^F; only the output is slightly modified to make it easier to read.

```
type tree = Empty | Node of tree * tree | Leaf of int
(* left-to-right traversal of the tree. *)
let rec (tree_iter: tree → graph) tree f accu  =
  begin match tree with
  | Empty → accu
  | Node (left, right) → tree_iter right f (tree_iter left f accu)
  | Leaf x → f x accu
  end
 : tree → graph
```

The annotation on `tree_iter` could be omitted here. It makes the inferred type more readable.

Next, we define a few generic functions that can be applied on any data structure, using the iterator.

```
let size  gr = gr (fun x n → n+1) 0     (* Number of elements. *)
let nodes gr = gr (fun x l → x::l) []   (* List of elements. *)
let sum   gr = gr (fun x s → s+x) 0     (* Sum of all elements. *)
let last  gr = gr (fun x a → x) 0       (* Last element, 0 if none. *)
let max   gr = gr (fun x m → if x > m then x else m) 0
let min   gr = gr (fun x m → if x < m then x else m) (last gr)
```

Since no type annotation is given, the inferred types may be quite large; for example:

```
val last : ∀(α ≥ ∀β. ∀(γ ≥ ∀δ. δ → β) β → γ) ∀ϵ. (α → int → ϵ) → ϵ
val max: ∀α. ((int → int → int) → int → α) → α
```

The following function converts a given graph (whose actual representation is unknown) to a list, provided the given graph is quite small. The list is exported as another graph.

```
let convert (gr:graph) =
  if size gr < 10 then list_iter (nodes gr)
  else gr
 : graph → graph
```

An annotation is required because the argument `gr` is used twice with incompatible types; that is, `gr` is used polymorphically. The inferred type is also much easier to read than the above types.

The next function (inefficiently) inserts an element in a balanced tree (usual ML code).

```
let rec insert tree x =
  begin match tree with
  | Empty → Leaf x
  | Leaf y → Node (Leaf x, Leaf y)
  | Node (left, right) →
      if size (tree_iter left) <= size (tree_iter right)
      then Node (insert left x, right)
      else Node (left, insert right x)
  end
 : tree → int → tree
```

Next, we build a balanced tree with the values 1, .., 6.

```
let tree = insert Empty 1
let tree = insert tree 2
let tree = insert tree 3
let tree = insert tree 4
let tree = insert tree 5
let tree = insert tree 6
```

We define three different graphs. The first one is implemented by a balanced tree; the second one is a conversion of it (and will actually be converted to a list); the third one is implemented by a list.

```
let graph₁ = tree_iter tree
let graph₂ = convert graph₁
let graph₃ = list_iter [1; 2; 3; 4; 5; 6]
```

The function `print_info` below prints all kinds of information about a given graph, including its size and all its nodes. It is a generic function: it does not depend on the underlying implementation.

```
 (* Tool function *)
let rec print_list l =
  if l = [] then print "[]"
  else (print_int (car l) ; print "; " ; print_list (cdr l))
```

```
let print_info (gr:graph) =
  begin
    print "Size  : " ; print_int (size gr) ; print "\n" ;
    print "Sum   : " ; print_int (sum  gr) ; print "\n" ;
    print "Last  : " ; print_int (last gr) ; print "\n" ;
    print "Max   : " ; print_int (max gr) ; print "\n" ;
    print "Min   : " ; print_int (min gr) ; print "\n" ;
    print "Nodes : " ; print_list (nodes gr) ; print "\n" ;
  end
 : graph → unit
```

This function is annotated because its argument `gr` is used polymorphically.

Finally, we call the generic function `print_info` with the three different graphs we have defined. Two of them are actually implemented by lists and one of them by a balanced tree.

```
print "Graph1 \n" ; print_info graph1 ;
print "Graph2 \n" ; print_info graph2 ;
print "Graph3 \n" ; print_info graph3 ;
```

The output we get is the following:

```
Graph1
Size  : 6
Sum   : 21
Last  : 1
Max   : 6
Min   : 1
Nodes : 1; 4; 6; 2; 3; 5; []

Graph2
Size  : 6
Sum   : 21
Last  : 5
Max   : 6
Min   : 1
Nodes : 5; 3; 2; 6; 4; 1; []
```

```
Graph3
Size  : 6
Sum   : 21
Last  : 6
Max   : 6
Min   : 1
Nodes : 6; 5; 4; 3; 2; 1; []
```

12.5 Discussion

In the above example, only two type annotations are mandatory, namely in `convert` and `print_info`. Another one was provided for convenience in `tree_iter`. Writing this sample code, it was always very clear when to put an annotation.

A different style would be to annotate generic functions such as `size`, `nodes`, etc., although they do not require any annotation. As a result, the inferred types would be nicer, and the function `print_info` would not need an annotation.

More generally, the "generic functions" above can be viewed as library functions. Then, their type would be provided in a signature file. As a consequence, user-defined functions using this library would, in most cases, not need type annotations.

Conclusion

We have presented ML^{F}, which is an extension of ML with more expressive types that embed first-class polymorphism. A sound and complete unification algorithm for these types is provided; although they are second-order, unification is kept first-order. Indeed, polymorphism is never guessed, but only silently propagated. It can be introduced, as in ML, by the `let` construct, or by explicit type annotations. Then it is implicitly instantiated, just as in ML.

The typing rules of ML^{F} are the typing rules of ML, up to the richer types and the richer instance relation. As a consequence, the type inference algorithm of ML^{F} is similar to the type inference of ML and basically relies on unification. As in ML, any typable ML^{F} program admits a principal type, that however depends on the explicit type annotations. The similarity between ML^{F} and ML imply that all ML programs are typable in ML^{F} as such. Furthermore, the richer types allow the introduction of type annotations as primitives, for which we prove type safety. Using these primitives, we show how to encode all System F programs by simply erasing type abstractions and type applications.

Interestingly, all type annotations of a System F program are not necessary in ML^{F}: monotype annotations as well as annotations on variables that are not used polymorphically can be inferred. For example, the `app` function, defined as $\lambda(f)\ \lambda(x)\ f\ x$ can be applied to any pair of arguments a_1 and a_2 such that $a_1\ a_2$ is typable. This is not true in System F where type applications are needed to specialize the type of `app`. Similarly, the function $\lambda(x)\ x\ \mathtt{id}$ can be applied both to a function expecting a monomorphic argument, such as `List.map`, and to a function expecting a polymorphic argument, such as `auto` (defined as $\lambda(x : \sigma_{\mathtt{id}})\ x\ x$). No type application or coercion is needed to choose between a monomorphic instance of the identity and the "polymorphic" identity. This is not true in other type systems trying to merge ML and first-class polymorphism, including Poly-ML.

In summary, ML^{F} is an integration of ML and System F that combines the convenience of type inference as present in ML and the expressiveness of second-order polymorphism. Type information is only required for arguments of functions that are used polymorphically in their bodies. We think that this specification should be intuitive to the user. Besides, it is modular, since annotations depend more on the behavior of the code than on the context in which the code is placed; in particular, functions that only carry polymorphism without using it can be left unannotated.

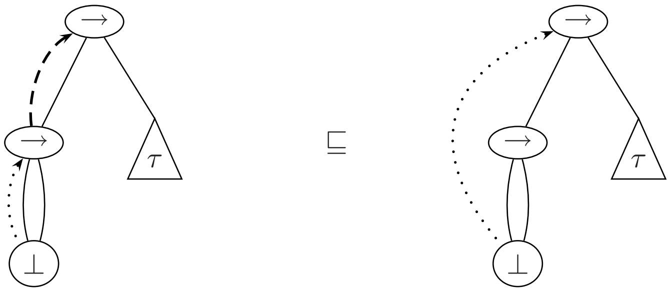
The obvious potential application of our work is to extend ML-like languages with second-order polymorphism while keeping full type inference for a large subset of the language, containing at least all ML programs.

Exploring the difference between ML^{F} and System F, we have considered a restriction of ML^{F} called Shallow ML^{F}. It is shown that Shallow ML^{F} can be equivalently viewed as ML^{F} where type annotations are restricted to types of System F. All programs of ML and of System F are typable in Shallow ML^{F} as well. This means that the programmer does not need to understand ML^{F} types to write Shallow ML^{F} programs. However, shallow types are still introduced by the type inference algorithm, *e.g.* the typing of `choose id` introduces the type $\forall\,(\alpha \geq \sigma_{\mathtt{id}})\ \alpha \to \alpha$. Hence, the knowledge of shallow types seems necessary to understand inferred types, in particular those appearing in error messages. Still, we have shown that Shallow ML^{F} (without the `let` construct) types the same set of terms than System F. More precisely, a typing of an expression in Shallow ML^{F} corresponds to one or more typings in System F. Thus, we expect that it is possible to design a typechecker based on ML^{F} that totally hides ML^{F} types: the programmer only sees System F types. However, although such a typechecker is able to infer principal types for expressions, it cannot always display them. This can also be problematic when writing interface files. Hence, whereas ML^{F} types are *a priori* not needed in source programs, they may still appear in interface files as well as in messages of the typechecker.

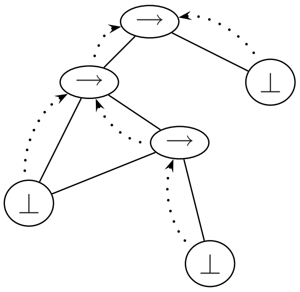
Future work

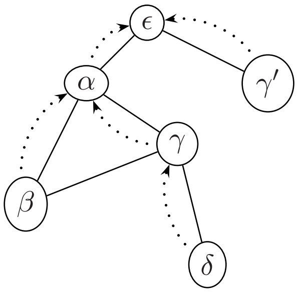
ML^{F} and F^{ω} Since System F can be encoded into ML^{F}, we may consider an extension of ML^{F} with higher-order types, that is, with types of F^{ω}. By keeping type abstractions and type applications fully explicit for higher-order kinds, it should be possible to perform type inference. As a consequence, we expect to be able to infer type abstractions and type applications for the kind `type` in F^{ω}. This remains to be formalized.

An efficient implementation of unification The unification algorithm we give is defined on "syntactic" types. Viewing the types as graphs, as described page 43, it should be possible to design a more efficient unification algorithm. More precisely, graphs and unification problems can be encoded as multi-equations, so that solving a unification problems amounts to finding the solved form of a set of multi-equations. Then the unification algorithm can be described as rewriting rules that keep the same set of solutions. From an implementation point of view, a unification algorithm on graphs would proceed in two steps: a first step performs first-order unification, ignoring binders; a second-step moves and merges binders. The second step fails if a rigid binder is illegally moved. For instance, rewriting the type $\forall\,(\alpha = \sigma_{\texttt{id}})\ \alpha \to \tau$ into $\forall\,(\beta)\ \forall\,(\alpha = \beta \to \beta)\ \alpha \to \tau$ amounts to moving the binder of β, which is not allowed under rigid bindings. Indeed, the former type requires a polymorphic argument, with type $\sigma_{\texttt{id}}$, whereas the latter can be applied to any monomorphic type. Therefore, such an instantiation is not correct because it would allow us to apply `auto`, which expects a polymorphic argument, to a monomorphic value such as `succ` (the successor function). On graphs, it means that the following is not allowed:

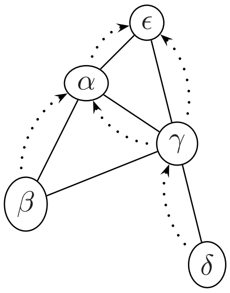


Additionally, some well-formedness conditions imply that each node must appear only under the node where it is bound. In "syntactic types", it corresponds to the fact that a type variable cannot be used outside its lexical scope. Thus, special care must be taken while merging binders: in order to keep well-formedness, it may be necessary to move binders not directly involved in the operation. Detecting which binders to move may be a costly operation; this needs further investigation. To illustrate this, we provide an example. The left-hand graph represents the type $\forall\,(\alpha \geq \forall\,(\beta)\ \forall\,(\gamma \geq \forall\,(\delta)\ \beta \to \delta)\ \beta \to \gamma)\ \forall\,(\gamma')\ \alpha \to \gamma'$. The right-hand graph is similar, but nodes are labelled with their corresponding variable name. The root node does not correspond to a type variable, thus we call it ϵ.

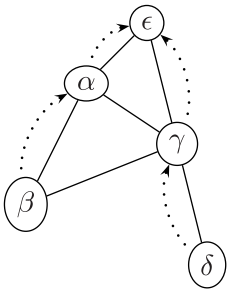




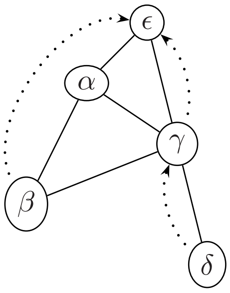
We wish to unify γ and γ'. The first step of the unification algorithm on graphs just performs first-order unification on structures and leads to the following:



This graph does not correspond to a syntactic type because γ is bound at two places. Fortunately, the binding of γ and the binding of α are flexible, thus it is allowed to move the binder of γ to the top-level node, which gives the following:



Unfortunately, this graph is not well-formed either: it is not possible to write it as a type. The reason is that β appears in the bound of γ but is bound at α. However, γ is bound at ϵ, thus β is not in the scope of γ. As a consequence, the binder of β needs to be moved to ϵ as follows:



This graph represents the type $\forall\,(\beta)\ \forall\,(\gamma \geq \forall\,(\delta)\ \beta \to \delta)\ (\beta \to \gamma) \to \gamma$, which is the solution to the unification problem described above.

This example illustrates that solving the unification of two nodes such as γ and γ' may have consequences on binders not directly related to γ or γ': in this example, it is necessary to move the binder of β to build a well-formed graph. Whereas algorithms for first-order unification are well-known and algorithms for merging binders seem straightforward, it remains to find an efficient algorithm to detect ill-formed graphs and to find which binders to move. This is probably the most costly operation since moving a single binder may transform arbitrarily many binders into illegal binders.

A semantics for ML$^{\mathbf{F}}$ types The interpretation of shallow types as sets of System F types sketched a semantics for ML$^{\mathrm{F}}$ types. However, as mentioned then, the given interpretation is not sound for the abstraction relation. Thus, it remains to find a suitable semantics that captures the meaning of ML$^{\mathrm{F}}$ types, including the meaning of abstraction and revelation. Indeed, the relations $\sqsubseteq$ and $\sqsupseteq$ are fundamental for type inference, and a semantics for types must be able to distinguish for example $\forall\,(\alpha = \sigma_{\mathtt{id}})\ \alpha \to \alpha$ and $\forall\,(\alpha = \sigma_{\mathtt{id}}, \beta = \sigma_{\mathtt{id}})\ \alpha \to \beta$. A starting point for such a semantics might be graph representations.

Bibliography

[Boe85] H.-J. Boehm. Partial polymorphic type inference is undecidable. In *26th Annual Symposium on Foundations of Computer Science*, pages 339–345. IEEE Computer Society Press, October 1985.

[BOSW98] Gilad Bracha, Martin Odersky, David Stoutamire, and Philip Wadler. Making the future safe for the past: Adding genericity to the java programming language. In *Proc. OPPSLA'98*, October 1998.

[Bou59] Nicolas Bourbaki. *Livre XI, Eléments de Mathématique, Algèbre, Chap 4: Polynomes et Fractions Rationnelles.* Hermann, 1959.

[Car88] Luca Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76(2–4):138–164, 1988.

[Car93] Luca Cardelli. An implementation of FSub. Research Report 97, Digital Equipment Corporation Systems Research Center, 1993.

[CC90] Felice Cardone and Mario Coppo. Two extensions of Curry's type inference system. In *Logic and Computer Science*, pages 19–76. Academic Press, 1990.

[CDC78] Mario Coppo and Mariangiola Dezani-Ciancaglini. A new type-assignment for λ-terms. *Archiv Math. Logik*, 19:139–156, 1978.

[CDCS79] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Patrick Sallé. Functional characterization of some semantic equalities inside λ-calculus. In Hermann A. Maurer, editor, *Proceedings of the 6th Colloquium on Automata, Languages and Programming*, volume 71 of *LNCS*, pages 133–146, Graz, Austria, July 1979. Springer.

[Cos95] Roberto Di Cosmo. *Isomorphisms of Types: from lambda-calculus to information retrieval and language design.* Birkhauser, 1995.

[DHKP96] Gilles Dowek, Thérèse Hardin, Claude Kirchner, and Frank Pfenning. Higher-order unification via explicit substitutions: the case of higher-order patterns. In M. Maher, editor, *Joint international conference and symposium on logic programming*, pages 259–273, 1996.

[DM82] Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *Proceedings of the Ninth ACM Conference on Principles of Programming Langages*, pages 207–212, 1982.

[Gar02] Jacques Garrigue. Relaxing the value-restriction. Presented at the third Asian workshop on Programmaming Languages and Systems (APLAS), 2002.

[Gir72] Jean-Yves Girard. *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur.* Thèse d'état, University of Paris VII, 1972.

[GR88] P. Giannini and S. Ronchi Della Rocca. Characterization of typings in polymorphic type discipline. In *Third annual Symposium on Logic in Computer Science*, pages 61–70. IEEE, 1988.

[GR99] Jacques Garrigue and Didier Rémy. Extending ML with Semi-Explicit higher-order polymorphism. *Journal of Functional Programming, vol 155*, pages pages 134–169, 1999. A preliminary version appeared in TACS'97.

[Hen93] Fritz Henglein. Type inference with polymorphic recursion. *ACM Transactions on Programming Languages and Systems*, 15(2):253–289, 1993.

[HP99] Haruo Hosoya and Benjamin C. Pierce. How good is local type inference? Technical Report MS-CIS-99-17, University of Pennsylvania, June 1999.

[Jim95] Trevor Jim. Rank-2 type systems and recursive definitions. Technical Report MIT/LCS/TM-531, Massachusetts Institute of Technology, Laboratory for Computer Science, November 1995.

[Jim96] Trevor Jim. What are principal typings and what are they good for? In ACM, editor, *ACM Symposium on Principles of Programming Languages (POPL), St. Petersburg Beach, Florida*, pages 42–53, 1996.

[Jim00] Trevor Jim. A polar type system. In *Electronic Notes in Theoretical Computer Science*, 2000.

[JS04] S. Peyton Jones and M. Shields. Practical type inference for arbitrary-rank types, 2004.

[JWOG89] Jr. James William O'Toole and David K. Gifford. Type reconstruction with first-class polymorphic values. In *SIGPLAN '89 Conference on Programming Language Design and Implementation*, Portland, Oregon, June 1989. ACM. also in ACM SIGPLAN Notices 24(7), July 1989.

[KW94] Assaf J. Kfoury and Joe B. Wells. A direct algorithm for type inference in the rank-2 fragment of the second-order lambda -calculus. In *ACM Conference on LISP and Functional Programming*, 1994.

[KW99] Assaf J. Kfoury and Joe B. Wells. Principality and decidable type inference for finite-rank intersection types. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 161–174. ACM, January 1999.

[LDG$^+$02] Xavier Leroy, Damien Doligez, Jacques Garrigue, Didier Rémy, and Jérôme Vouillon. The Objective Caml system, documentation and user's manual - release 3.05. Technical report, INRIA, July 2002. Documentation distributed with the Objective Caml system.

[LO94a] Konstantin Läufer and Martin Odersky. Polymorphic type inference and abstract data types. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1411–1430, September 1994. An earlier version appeared in the Proceedings of the ACM SIGPLAN Workshop on ML and its Applications, 1992, under the title "An Extension of ML with First-Class Abstract Types".

[LO94b] Konstantin Läufer and Martin Odersky. Polymorphic type inference and abstract data types. *ACM Transactions on Programming Languages and Systems*, 16(5):1411–1430, September 1994.

[Mil78] R. Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17:348–375, December 1978.

[Mil92] Dale Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 14:321–358, 1992.

[Mit83] John C. Mitchel. Coercion and type inference. In *ACM Symp. on Priciples of Programming Languages*, pages 175–185. ACM, 1983.

[Mit88] John C. Mitchell. Polymorphic type inference and containment. *Information and Computation*, 2/3(76):211–249, 1988.

[Oka98] Chris Okasaki. *Purely Functional Data Structures.* Cambridge University Press, Cambridge, England, 1998.

[OL96] Martin Odersky and Konstantin Läufer. Putting type annotations to work. In *Proceedings of the 23rd ACM Conference on Principles of Programming Languages*, pages 54–67, January 1996.

[OSW99] Martin Odersky, Martin Sulzmann, and Martin Wehr. Type inference with constrained types. *Theory and Practice of Object Systems*, 5(1):35–55, 1999.

[OZZ01] Martin Odersky, Christoph Zenger, and Matthias Zenger. Colored local type inference. *ACM SIGPLAN Notices*, 36(3):41–53, March 2001.

[Pfe88] Frank Pfenning. Partial polymorphic type inference and higher-order unification. In *Proceedings of the ACM Conference on Lisp and Functional Programming*, pages 153–163. ACM Press, July 1988.

[Pfe93] Frank Pfenning. On the undecidability of partial polymorphic type reconstruction. *Fundamenta Informaticae*, 19(1,2):185–199, 1993. Preliminary version available as Technical Report CMU-CS-92-105, School of Computer Science, Carnegie Mellon University, January 1992.

[Pie91] Benjamin C. Pierce. *Programming with Intersection Types and Bounded Polymorphism*. PhD thesis, Carnegie Mellon University, December 1991. Available as School of Computer Science technical report CMU-CS-91-205.

[Pot80] Garrell Pottinger. A type assignment for the strongly normalizable λ-terms. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 561–577. Academic Press, New York, 1980.

[Pot98a] François Pottier. A framework for type inference with subtyping. In *Proceedings of the third ACM SIGPLAN International Conference on Functional Programming (ICFP'98)*, pages 228–238, September 1998.

[Pot98b] François Pottier. *Synthèse de types en présence de sous-typage: de la théorie à la pratique*. PhD thesis, Université Paris 7, July 1998.

[PT98] Benjamin C. Pierce and David N. Turner. Local type inference. In *Proceedings of the 25th ACM Conference on Principles of Programming Languages*, 1998. Full version in *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 22(1), January 2000, pp. 1–44.

[Rém92] Didier Rémy. Extending ML type system with a sorted equational theory. Research Report 1766, Institut National de Recherche en Informatique et Automatisme, Rocquencourt, BP 105, 78 153 Le Chesnay Cedex, France, 1992.

[Rém94] Didier Rémy. Programming objects with ML-ART: An extension to ML with abstract and record types. In Masami Hagiya and John C. Mitchell, editors, *Theoretical Aspects of Computer Software*, volume 789 of *Lecture Notes in Computer Science*, pages 321–346. Springer-Verlag, April 1994.

[Rey74] John C. Reynolds. Towards a theory of type structure. In *Proc. Colloque sur la Programmation*, pages 408–425, New York, 1974. Springer-Verlag LNCS 19.

[Sal82] P. Sallé. Une extension de la theorie des types en λ-calcul. In *Lecture Notes in Computer Science No. 62*, pages 398–410. Springer-Verlag, 1982.

[Sch98] Aleksy Schubert. Second-order unification and type inference for Church-style polymorphism. In *Conference Record of POPL 98: The 25TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, California*, pages 279–288, New York, NY, 1998.

[Sim03] Vincent Simonet. The Flow Caml System: documentation and user's manual. Technical Report 0282, Institut National de Recherche en Informatique et en Automatique (INRIA), July 2003. ©INRIA.

[Wel94] J. B. Wells. Typability and type checking in the second order λ-calculus are equivalent and undecidable. In *Ninth annual IEEE Symposium on Logic in Computer Science*, pages 176–185, July 1994.

[Wel96] Joe B. Wells. *Type Inference for System F with and without the Eta Rule*. PhD thesis, Boston University, 1996.

[Wel02] J. B. Wells. The essence of principal typings. In *Proc. 29th Int'l Coll. Automata, Languages, and Programming*, volume 2380 of *LNCS*, pages 913–925. Springer-Verlag, 2002.

[Wri95] Andrew K. Wright. Simple imperative polymorphism. *Lisp and Symbolic Computation*, 8(4):343–355, 1995.

Part IV

Appendix

Appendix A

Proofs (Technical details)

Felix qui potuit rerum cognescere causas — Virgile Géorgiques II, 489

Proof of Property 1.5.6

Property i: It is by induction on the number of universal quantifiers appearing in σ. If σ has no universal quantifiers, then σ is $\bot$ or τ, and we get the result by Rule Eq-Refl since $\mathsf{nf}(\sigma)$ is σ itself. Otherwise, σ is $\forall\,(\alpha \diamond \sigma_1)\ \sigma_2$, where σ_1 and σ_2 have strictly less universal quantifiers than σ. We proceed by case analysis:

- Case $\mathsf{nf}(\sigma_2)$ is α: Then $\mathsf{nf}(\sigma)$ is $\mathsf{nf}(\sigma_1)$ **(1)**, and $\sigma_2 \equiv \alpha$ by induction hypothesis. By Property 1.5.3.v (page 49), $(\alpha \diamond \sigma_1)\ \sigma_2 \equiv \alpha$ also holds. Hence, $\sigma \equiv \sigma_1$ **(2)** holds by rules R-Context-R and Eq-Var. By induction hypothesis on σ_1, we have $\sigma_1 \equiv \mathsf{nf}(\sigma_1)$ **(3)**, therefore $\sigma \equiv \mathsf{nf}(\sigma_1)$ holds by R-Trans, (2), and (3). By (1), we get $\sigma \equiv \mathsf{nf}(\sigma)$.
- Case $\alpha \notin \mathsf{ftv}(\sigma_2)$: then, $\mathsf{nf}(\sigma)$ is $\mathsf{nf}(\sigma_2)$ **(4)**. We have $\sigma \equiv \sigma_2$ by Eq-Free, thus $\sigma \equiv \mathsf{nf}(\sigma_2)$ holds by induction hypothesis and R-Trans. By (4), this means $\sigma \equiv \mathsf{nf}(\sigma)$.
- Case $\mathsf{nf}(\sigma_1)$ is τ_1: We have $\sigma_2 \equiv \mathsf{nf}(\sigma_2)$ by induction hypothesis on σ_2, thus $\sigma \equiv \forall\,(\alpha \diamond \sigma_1)\ \mathsf{nf}(\sigma_2)$ **(5)** holds by R-Context-R and Property 1.5.3.v (page 49). Moreover, we have $\sigma_1 \equiv \tau_1$ by induction hypothesis on σ_1, hence we have $\sigma \equiv \mathsf{nf}(\sigma_2)[\tau_1/\alpha]$ by Rule Eq-Mono* on (5) and R-Trans. We conclude by observing that $\mathsf{nf}(\sigma)$ is $\mathsf{nf}(\sigma_2)[\tau_1/\alpha]$.
- Otherwise, $\mathsf{nf}(\sigma)$ is $\forall\,(\alpha \diamond \mathsf{nf}(\sigma_1))\ \mathsf{nf}(\sigma_2)$. By induction hypothesis, $\sigma_1 \equiv \mathsf{nf}(\sigma_1)$ and $\sigma_2 \equiv \mathsf{nf}(\sigma_2)$. Hence, $(\alpha \diamond \sigma_1)\ \sigma_2 \equiv \mathsf{nf}(\sigma_2)$ holds by Property 1.5.3.v (page 49). We conclude by rules R-Context-R, R-Context-R and R-Trans.

Property ii: It is a consequence of Properties i and 1.5.4.iii (page 50).

Property iii: It is shown by induction on the number of universal quantifiers appearing in σ. If σ has no universal quantifiers, then σ is $\bot$ or τ, and we get the expected result since $\mathsf{nf}(\sigma)$ is σ itself and $\mathsf{nf}(\theta(\sigma))$ is $\theta(\sigma)$. Otherwise, σ is $\forall\,(\alpha \diamond \sigma_1)\ \sigma_2$, where σ_1 and σ_2 have strictly less universal quantifiers than σ. By alpha-conversion, we can assume $\alpha \notin \mathsf{dom}(\theta) \cup \mathsf{codom}(\theta)$, thus $\theta(\sigma) = \forall\,(\alpha \diamond \theta(\sigma_1))\ \theta(\sigma_2)$. We proceed by case analysis:

∘ CASE $\mathsf{nf}(\sigma_2)$ is α: By induction hypothesis, $\mathsf{nf}(\theta(\sigma_2))$ is $\theta(\mathsf{nf}(\sigma_2))$, that is α. Hence, $\mathsf{nf}(\theta(\sigma))$, that is, $\mathsf{nf}(\forall\,(\alpha \diamond \theta(\sigma_1))\ \theta(\sigma_2))$, is by definition $\mathsf{nf}(\theta(\sigma_1))$. By induction hypothesis, $\mathsf{nf}(\theta(\sigma_1))$ is $\theta(\mathsf{nf}(\sigma_1))$, thus $\mathsf{nf}(\theta(\sigma))$ is $\theta(\mathsf{nf}(\sigma_1))$, that is, $\theta(\mathsf{nf}(\sigma))$.

∘ CASE $\mathsf{nf}(\sigma_1)$ is τ_1: Let θ' be $[\tau_1/\alpha]$ and θ'' be $[\theta(\tau_1)/\alpha]$; since $\alpha \notin \mathsf{dom}(\theta) \cup \mathsf{codom}(\theta)$, we have $\theta \circ \theta' = \theta'' \circ \theta$ **(6)**. By induction hypothesis $\mathsf{nf}(\theta(\sigma_1))$ is $\theta(\tau_1)$, thus $\mathsf{nf}(\theta(\sigma))$ is by definition $\theta''(\mathsf{nf}(\theta(\sigma_2)))$. By induction hypothesis, it is also $\theta''(\theta(\mathsf{nf}(\sigma_2)))$, that is $\theta \circ \theta'(\mathsf{nf}(\sigma_2))$ by (6). We conclude by observing that $\theta'(\mathsf{nf}(\sigma_2))$ is $\mathsf{nf}(\sigma)$.

∘ CASE $\alpha \notin \mathsf{ftv}(\sigma_2)$: Since $\alpha \notin \mathsf{codom}(\theta)$, we have $\alpha \notin \mathsf{ftv}(\theta(\sigma_2))$. Hence, $\mathsf{nf}(\theta(\sigma))$ is by definition $\mathsf{nf}(\theta(\sigma_2))$ and we conclude directly by induction hypothesis.

∘ OTHERWISE, we know that $\mathsf{nf}(\sigma_2)$ is not α, thus, by induction hypothesis, $\mathsf{nf}(\theta(\sigma_2))$ is not α. Moreover, $\mathsf{nf}(\sigma_1)$ is not a monotype τ_1, thus, by induction hypothesis, $\mathsf{nf}(\theta(\sigma_1))$ is not a monotype. Additionally, $\alpha \in \mathsf{ftv}(\sigma_2)$, thus $\alpha \in \mathsf{ftv}(\theta(\sigma_2))$. Consequently, $\mathsf{nf}(\theta(\sigma))$ is by definition $\forall\,(\alpha \diamond \mathsf{nf}(\theta(\sigma_1)))\ \mathsf{nf}(\theta(\sigma_2))$. By induction hypothesis, it is equal to $\forall\,(\alpha \diamond \theta(\mathsf{nf}(\sigma_1)))\ \theta(\mathsf{nf}(\sigma_2))$, that is, $\theta(\mathsf{nf}(\sigma))$.

Property iv: It is shown by induction on the number of universal quantifiers appearing in σ. If σ has no universal quantifiers, then σ is $\bot$ or τ, and we get the expected result since $\mathsf{nf}(\sigma)$ is σ itself. Otherwise, σ is $\forall\,(\alpha \diamond \sigma_1)\ \sigma_2$, where σ_1 and σ_2 have strictly less universal quantifiers than σ. We proceed by case analysis:

∘ CASE $\mathsf{nf}(\sigma_2)$ is α: then, $\mathsf{nf}(\sigma)$ is $\mathsf{nf}(\sigma_1)$, which is in normal form by induction hypothesis.

∘ CASE $\mathsf{nf}(\sigma_1)$ is τ_1: let θ be $[\tau_1/\alpha]$; by definition, $\mathsf{nf}(\sigma)$ is $\theta(\mathsf{nf}(\sigma_2))$, that is, $\mathsf{nf}(\theta(\sigma_2))$ by Property iii. Since $\mathsf{nf}(\theta(\sigma_2))$ is in normal form by induction hypothesis, we conclude that $\mathsf{nf}(\sigma)$ is normal form.

∘ CASE $\alpha \notin \mathsf{ftv}(\sigma_2)$: then, $\mathsf{nf}(\sigma)$ is $\mathsf{nf}(\sigma_2)$, which is in normal form by induction hypothesis.

∘ OTHERWISE, $\mathsf{nf}(\sigma)$ is $\forall\,(\alpha \diamond \mathsf{nf}(\sigma_1))\ \mathsf{nf}(\sigma_2)$. We know that $\mathsf{nf}(\sigma_2)$ is not α, that $\mathsf{nf}(\sigma_1)$ is not a monotype τ_1, and that $\alpha \in \mathsf{ftv}(\sigma_2)$. By Property i, we have $\mathsf{nf}(\sigma_2) \equiv \sigma_2$. Hence, by Property 1.5.4.iii (page 50), we have $\alpha \in \mathsf{ftv}(\mathsf{nf}(\sigma_2))$. Additionally, by induction hypothesis, $\mathsf{nf}(\mathsf{nf}(\sigma_1))$ is $\mathsf{nf}(\sigma_1)$ and $\mathsf{nf}(\mathsf{nf}(\sigma_2))$ is $\mathsf{nf}(\sigma_2)$. Hence, $\mathsf{nf}(\mathsf{nf}(\sigma))$ is $\mathsf{nf}(\forall\,(\alpha \diamond \mathsf{nf}(\sigma_1))\ \mathsf{nf}(\sigma_2))$, that is by definition $\forall\,(\alpha \diamond \mathsf{nf}(\sigma_1))\ \mathsf{nf}(\sigma_2)$, that is $\mathsf{nf}(\sigma)$. Hence, $\mathsf{nf}(\sigma)$ is in normal form. ■

Proof of Corollary 1.5.10

Directly, if $(\emptyset)\ \widehat{Q}(\sigma_1) \equiv \widehat{Q}(\sigma_2)$ holds, then, by Property 1.5.3.v (page 49), we have $(Q)\ \widehat{Q}(\sigma_1) \equiv \widehat{Q}(\sigma_2)$. Since we have $(Q)\ \sigma_1 \equiv \widehat{Q}(\sigma_1)$ and $(Q)\ \sigma_2 \equiv \widehat{Q}(\sigma_2)$ by EQ-MONO, we conclude directly using R-TRANS twice. Conversely, if we have $(Q)\ \sigma_1 \equiv \sigma_2$, then by Lemma 1.5.9 we know that $\mathsf{nf}(\widehat{Q}(\sigma_1)) \approx \mathsf{nf}(\widehat{Q}(\sigma_2))$ holds. Hence, $\mathsf{nf}(\widehat{Q}(\sigma_1)) \equiv \mathsf{nf}(\widehat{Q}(\sigma_2))$ holds by Property 1.5.3.iii (page 49). We conclude by observing that $\widehat{Q}(\sigma_1) \equiv \mathsf{nf}(\widehat{Q}(\sigma_1))$ and $\mathsf{nf}(\widehat{Q}(\sigma_2)) \equiv \widehat{Q}(\sigma_2)$ hold by Property 1.5.6.i (page 51). ■

Proof of Property 2.1.2

Property i: *Reflexivity:* immediate *Transitivity:* Assume we have $t_1 \leqslant_/ t_2$ $(\underline{\mathbf{1}})$ and $t_2 \leqslant_/ t_3$ $(\underline{\mathbf{2}})$. We have $\mathsf{dom}(t_1) \subseteq \mathsf{dom}(t_2) \subseteq \mathsf{dom}(t_3)$, thus $\mathsf{dom}(t_1) \subseteq \mathsf{dom}(t_3)$. Besides, for all $u \in \mathsf{dom}(t_1)$, if $t_1/u \neq t_3/u$, then either $t_1/u = t_2/u$, or $t_1/u \neq t_2/u$. In the second case, we have $t_1/u = \bot$ by definition of (1), which is the expected result. In the first case, we must have $t_2/u \neq t_3/u$, thus $t_2/u = \bot$ by definition of (2), and $t_1/u = t_2/u = \bot$ holds. *Antisymmetry:* Assume $t_1 \leqslant_/ t_2$ $(\underline{\mathbf{3}})$ and $t_2 \leqslant_/ t_1$ $(\underline{\mathbf{4}})$. We have $\mathsf{dom}(t_1) \subseteq \mathsf{dom}(t_2)$ and $\mathsf{dom}(t_2) \subseteq \mathsf{dom}(t_1)$, thus $\mathsf{dom}(t_1) = \mathsf{dom}(t_2)$ holds. Besides, for any u in $\mathsf{dom}(t_1)$, if we have $t_1/u \neq t_2/u$, then $t_1/u = \bot$ by definition of (3). However, we then also have $t_2/u \neq t_1/u$, which implies $t_2/u = \bot$ by definition of (4). Consequently, $t_1/u = \bot = t_2/u$, which is a contradiction. As a consequence, $t_1/ = t_2/$, that is, $t_1 = t_2$.

Property ii: Assume we have $t_1 \leqslant_/ t_2$ $(\underline{\mathbf{1}})$ By definition, we have $\mathsf{dom}(t_1) \subseteq \mathsf{dom}(t_2)$ $(\underline{\mathbf{2}})$. Let θ be a substitution. Note that $\mathsf{dom}(t) \subseteq \mathsf{dom}(\theta(t))$ $(\underline{\mathbf{3}})$ holds for any t. Let u be in $\mathsf{dom}(\theta(t_1))$. Either u is in $\mathsf{dom}(t_1)$, or u is of the form u_1u_2, with $t_1/u_1 = \alpha$ $(\underline{\mathbf{4}})$ and $\theta(\alpha)/u_2 = t_1/u$. In the first case, we know that u is in $\mathsf{dom}(t_2)$ from (2). Hence, u is in $\mathsf{dom}(\theta(t_2))$ by (3). In the second case, we know that u_1 is in $\mathsf{dom}(t_2)$ from (2), and that t_2/u_1 is α by definition of (1) and by (4). Hence, u_1u_2 is in $\mathsf{dom}(\theta(t_2))$. Consequently, we have shown that $\mathsf{dom}(\theta(t_1)) \subseteq \mathsf{dom}(\theta(t_2))$ holds $(\underline{\mathbf{5}})$. Let u be such that $\theta(t_1)/u \neq \theta(t_2)/u$ $(\underline{\mathbf{6}})$. Either u is in $\mathsf{dom}(t_1)$, or u is of the form u_1u_2, with $t_1/u_1 = \alpha$ and $\theta(\alpha)/u_2 = t_1/u$. In the first case, u is in $\mathsf{dom}(t_2)$ by (2), and $t_1/u \neq t_2/u$, which implies $t_1/u = \bot$ by definition of (1). Hence, $\theta(t_1)/u = \bot$, since $\bot$ is not substituted by θ. In the second case, $t_1/u_1 = \alpha$, thus $t_2/u_2 = \alpha$ by definition of (1). Hence, we have $\theta(t_1)/u_1u_2 = \theta(\alpha)/u_2 = \theta(t_2)/u_1u_2$, that is $\theta(t_1)/u = \theta(t_2)/u$. This is a contradiction with (6). Hence, this second case cannot occur. In summary, we have shown that whenever $\theta(t_1)/u \neq \theta(t_2)/u$, then $\theta(t_1)/u = \bot$. With (5), this implies that we have $\theta(t_1) \leqslant_/ \theta(t_2)$.

Property iii: We assume we have $t_1 \leqslant_/ t_2$ $(\underline{\mathbf{1}})$ and a skeleton t. By definition, $\mathsf{dom}(t_1) \subseteq \mathsf{dom}(t_2)$ $(\underline{\mathbf{2}})$ holds. Let t'_1 be $t[t_1/\alpha]$ and t'_2 be $t[t_2/\alpha]$. Note that we have $\mathsf{dom}(\tau) \subseteq \mathsf{dom}(t_1)$ and $\mathsf{dom}(t) \subseteq \mathsf{dom}(t_2)$ $(\underline{\mathbf{3}})$. We have to show that $t'_1 \leqslant_/ t'_2$ holds. Let u be in $\mathsf{dom}(t'_1)$. Either u is in $\mathsf{dom}(t)$, or u is of the form u_1u_2 with $t/u_1 = \alpha$ $(\underline{\mathbf{4}})$ and $t_1/u_2 = t'_1/u$. In the first case, u is in $\mathsf{dom}(t'_2)$ by (3). In the second case, we have $u_2 \in \mathsf{dom}(t_1)$, hence $u_2 \in \mathsf{dom}(t_2)$ $(\underline{\mathbf{5}})$ by (2). Then u_1u_2 is in $\mathsf{dom}(t[t_2/\alpha])$ by (4) and (5). We have shown that $\mathsf{dom}(t'_1) \subseteq \mathsf{dom}(t'_2)$ $(\underline{\mathbf{6}})$. Let u be such that $t'_1/u \neq t'_2/u$ $(\underline{\mathbf{7}})$. Either u is in $\mathsf{dom}(t)$, or u is of the form u_1u_2 with $t/u_1 = \alpha$ and $t_1/u_2 = t'_1/u$. In the first case, we necessarily have $t/u = \alpha$ (otherwise, $t'_1/u = t/u = t'_2/u$). Hence, $t'_1/u = t_1/\epsilon$ and $t'_2/u = t_2/\epsilon$, thus we have $t_1/\epsilon \neq t_2/\epsilon$ from (7), which implies $t_1/\epsilon = \bot$ by definition of (1). This leads to $t'_1/u = \bot$. In the second case, we have $t'_1/u = t_1/u_2$

and $t'_2/u = t_2/u_2$. Hence, $t_1/u_2 \neq t_2/u_2$ holds from (7), which implies $t_1/u_2 = \bot$ by definition of (1). This leads to $t'_1/u = \bot$. In both cases, $t'_1/u = \bot$. With (6), this implies that we have $t'_1 \leqslant_{/} t'_2$. ■

Proof of Lemma 2.1.4

By induction on the derivation of $(Q, \alpha \diamond \sigma, Q_1)\ \sigma_1 \Diamond \sigma_2$. By hypothesis, we have $(\forall\,(Q_1)\ \sigma_1)/u = \alpha$ $\mathbf{(\underline{1})}$.

∘ Case A-Equiv: we have $(Q, \alpha \diamond \sigma, Q_1)\ \sigma_1 \equiv \sigma_2$, thus $(Q, \alpha \diamond \sigma)\ \forall\,(Q_1)\ \sigma_1 \equiv \forall\,(Q_1)\ \sigma_2$ $\mathbf{(\underline{2})}$ holds by Rule R-Context-R. Let θ be $\widehat{(Q, \alpha \diamond \sigma)}$. Since $\sigma \notin \mathcal{T}$, we have $\alpha \notin \mathsf{dom}(\theta)$. By well-formedness of $(Q, \alpha \diamond \sigma)$, we have Q closed and $\alpha \notin \mathsf{dom}(Q)$, thus $\alpha \notin \mathsf{codom}(\theta)$ $\mathbf{(\underline{3})}$. By Corollary 1.5.10 applied to (2), we have $\theta(\forall\,(Q_1)\ \sigma_1) \equiv \theta(\forall\,(Q_1)\ \sigma_2)$. By Property 1.5.11.vi (page 54), we have $\theta(\forall\,(Q_1)\ \sigma_1)/ = \theta(\forall\,(Q_1)\ \sigma_2)/$ $\mathbf{(\underline{4})}$. By (1), we have $\theta(\forall\,(Q_1)\ \sigma_1)/u = \theta(\alpha)/\epsilon = \alpha$ and $\theta(\forall\,(Q_1)\ \sigma_2)/u = \alpha$ by (4), which implies $(\forall\,(Q_1)\ \sigma_2)/u = \alpha$, by (3).

∘ Case R-Trans: By induction hypothesis.

∘ Case R-Trans: By induction hypothesis.

∘ Case I-Hyp: we have $(Q, \alpha \diamond \sigma, Q_1)\ \sigma_1 \sqsubseteq \beta$ with $(\beta \diamond' \sigma_1) \in (Q, \alpha \diamond \sigma, Q_1)$. If $(\beta \diamond' \sigma_1) \in (Q, \alpha \diamond \sigma)$, necessarily $\mathsf{ftv}(\sigma_1)\ \#\ \{\alpha\} \cup \mathsf{dom}(Q_1)$ by well-formedness of $(Q, \alpha \diamond \sigma, Q_1)$, thus $\forall\,(Q_1)\ \sigma_1 \equiv \sigma_1$ (by Eq-Free) and $\alpha \notin \mathsf{ftv}(\forall\,(Q_1)\ \sigma_1)$, which is a contradiction with (1). Hence, $\beta \in \mathsf{dom}(Q_1)$. Therefore, $\forall\,(Q_1)\ \sigma_1 \equiv \forall\,(Q_1)\ \beta$ by Eq-Var*. Consequently, $(\forall\,(Q_1)\ \beta)/u = (\forall\,(Q_1)\ \sigma_1)/u = \alpha$ by Property 1.5.4.i (page 50) and (1).

∘ Case A-Hyp: similar to I-Hyp.

∘ Case R-Context-R: By induction hypothesis.

∘ Case R-Context-R: By induction hypothesis.

∘ Case R-Context-Flexible: We have $(Q, \alpha \diamond \sigma, Q_1)\ \forall\,(\beta \diamond \sigma_1)\ \sigma_0 \Diamond \forall\,(\beta \diamond \sigma_2)\ \sigma_0$ and the premise is $(Q, \alpha \diamond \sigma, Q_1)\ \sigma_1 \Diamond \sigma_2$ $\mathbf{(\underline{5})}$. Besides, $(\forall\,(Q_1)\ \forall\,(\beta \diamond \sigma_1)\ \sigma_0)/u = \alpha$ by hypothesis (1). By Property 1.3.3.i (page 40), we have $\Theta_{Q_1}(\mathsf{proj}(\sigma_0)[\mathsf{proj}(\sigma_1)/\alpha])/u = \alpha$. If u is of the form u_1u_2 with $\sigma_0/u_1 = \beta$, then $\Theta_{Q_1}(\mathsf{proj}(\sigma_1))/u_2 = \alpha$, thus $\forall\,(Q_1)\ \sigma_1/u_2 = \alpha$ by Property 1.3.3.i (page 40). Hence, by induction hypothesis on (5), we get $(\forall\,(Q_1)\ \sigma_2)/u_2 = \alpha$, which implies $(\forall\,(Q_1)\ \forall\,(\beta \diamond \sigma_2)\ \sigma_0)/u_1u_2 = \alpha$ by Property 1.3.3.i (page 40). Otherwise, we have $(\forall\,(Q_1)\ \forall\,(\beta \diamond \sigma_1)\ \sigma_0)/u = (\forall\,(Q_1)\ \sigma_0)/u = (\forall\,(Q_1)\ \forall\,(\beta \diamond \sigma_2)\ \sigma_0)/u$, which is the expected result.

∘ Case R-Context-Rigid is similar to R-Context-Flexible.

∘ Case I-Abstract: By induction hypothesis.

∘ Case I-Bot cannot occur since $(\forall\,(Q_1)\ \bot)/u = \alpha$ is not possible.

∘ Case I-Rigid is immediate. ■

Proof of Property 2.1.5

Property i: If there exists $u \in \mathsf{dom}(\sigma)$ such that $\sigma/u = \bot$, then by Property 1.5.4.i (page 50), σ is not equivalent to a type (no type τ is such that $\tau/u = \bot$). Hence, by Property 1.5.11.ii (page 54), σ is not in $\mathcal{T}$. Conversely, assume that σ is not equivalent to a type. We have to show that there exists u such that $\sigma/u = \bot$. The proof is by induction on the structure of σ. If σ is $\bot$, we get the result by taking $u = \epsilon$. Otherwise, σ is of the form $\forall\,(\alpha \diamond \sigma_1)\ \sigma_2$. If both σ_1 and σ_2 are equivalent to types τ_1 and τ_2, then by Rule Eq-Mono, σ would be equivalent to a monotype $\tau_2[\tau/\alpha]$, which is not possible by hypothesis. Hence, σ_1 or σ_2 is not equivalent to a type. If σ_2 is not equivalent to a type, then by induction hypothesis, there exists u such that $\sigma_2/u = \bot$, thus $\sigma/u = \bot$. If σ_2 is equivalent to a type, then necessarily σ_1 is not and $\alpha \in \mathsf{dom}(\sigma_2)$ (otherwise σ would be equivalent to σ_2 by Rule Eq-Free). Hence, there exists u_2 such that $\sigma_2/u_2 = \alpha$. Furthermore, by induction hypothesis, there exists u_1 such that $\sigma_1/u_1 = \bot$. Consequently, $\sigma/u_2u_1 = \bot$.

Property ii: Directly, by Property 1.5.4.i (page 50). Conversely, we assume σ/ϵ is α. Since α is of arity 0, we must have $\mathsf{dom}(\sigma) = \{\epsilon\}$. Hence, there is no $u \in \mathsf{dom}(\sigma)$ such that $\sigma/u = \bot$. By Property i, this implies $\sigma \in \mathcal{T}$. Consequently, $\mathsf{nf}(\sigma) = \tau$ such that $\tau/ = \sigma/ = \alpha/$, by Properties 1.5.6.i (page 51) and 1.5.4.i (page 50). This implies $\tau = \alpha$ (τ and α are viewed as skeletons). Hence, $\mathsf{nf}(\sigma) = \alpha$, that is, $\sigma \equiv \alpha$ by Property 1.5.6.i (page 51).

Property iii: It is shown by induction on the structure of σ. By hypothesis, we have $\sigma/\epsilon = \bot$ **(1)**. If σ is $\bot$, the result is immediate. Otherwise, by Property i and (1), we know that σ is not a type τ. If σ is $\forall\,(\alpha \diamond \sigma_1)\ \sigma_2$, either $\sigma_2/\epsilon = \bot$ and we conclude by induction hypothesis and Eq-Free, either $\sigma_2/\epsilon = \alpha$ and $\sigma_1/\epsilon = \bot$ **(2)**. In the latter case, we get $\sigma_1 \equiv \bot$ **(3)** by induction hypothesis on (2) and $\sigma_2 \equiv \alpha$ **(4)** by Property ii, thus we conclude that $\forall\,(\alpha \diamond \sigma_1)\ \sigma_2 \equiv \bot$ by R-Trans, R-Context-L, R-Context-R, Eq-Var, (3), and (4). This is the expected result. ■

Proof of Property 2.2.2

Property i : It is shown by induction on the size of Q.

Property ii: We show by induction on the size of Q that $Q[\alpha] = \beta$.

Property iii : It is shown by structural induction on Q. By hypothesis, we have $\widehat{Q}(\alpha) \notin \mathcal{V}$ **(1)**. If Q is $(Q', \beta \diamond \sigma)$, with $\beta \neq \alpha$, then we have $\alpha \in \mathsf{dom}(Q')$, $\widehat{Q'}(\alpha) = \widehat{Q}(\alpha)$, and $Q(\alpha) = Q'(\alpha)$, thus we get the result by induction hypothesis. If Q is $(Q', \alpha \diamond \sigma)$ and $\mathsf{nf}(\sigma) = \beta$, then we have $\beta \in \mathsf{dom}(Q')$ by well-formedness and $(Q)\ \alpha \equiv \beta$ **(2)** by

Property 1.5.6.i (page 51) and Eq-Mono. Additionally, we have $Q[\alpha] = Q'[\beta]$, thus $Q(\alpha) = Q'(\beta)$ $(\underline{\mathbf{3}})$ holds. Moreover, we have $\widehat{Q}(\alpha) = \widehat{Q'}(\beta)$, hence $\widehat{Q'}(\beta) \notin \mathcal{V}$ holds by (1). By induction hypothesis, $(Q')\ \beta \equiv Q'(\beta)$ holds. Hence, we have $(Q)\ \beta \equiv Q'(\beta)$ $(\underline{\mathbf{4}})$ by Property 1.5.3.v (page 49). Finally, $(Q)\ \alpha \equiv Q(\alpha)$ holds by (2), (4), and (3). This is the expected result. In the last case, Q is $(Q', \alpha \diamond \sigma)$, and $\sigma \notin \vartheta$. Then $Q[\alpha]$ is α, thus $Q(\alpha)$ is σ. Moreover, if $\sigma \notin \mathcal{T}$, then $\widehat{Q}(\alpha)$ is α, which is a contradiction. Consequently, $\sigma \in \mathcal{T}$, and $\widehat{Q}(\alpha) \equiv \sigma$ $(\underline{\mathbf{5}})$ by Definition 1.5.8 and Property 1.5.6.i (page 51). Hence, by Eq-Mono and (5), we get $(Q)\ \alpha \equiv \sigma$, that is, $(Q)\ \alpha \equiv Q(\alpha)$.

Property iv: It is shown by induction on the size of Q. Necessarily, Q is of the form $(Q_1, \alpha \diamond \sigma, Q_2)$. By definition, $Q[\alpha]$ is $Q_1[\beta]$ if $\mathsf{nf}(\sigma)$ is β and α otherwise $(\underline{\mathbf{1}})$. In the first case, $Q(\alpha)$ is $Q_1(\beta)$ by definition and $\widehat{Q}(\alpha)$ is $\widehat{Q_1}(\beta)$, hence we get the expected result by induction hypothesis on Q_1. In the second case, $Q(\alpha)$ is σ $(\underline{\mathbf{2}})$. Directly, we have by hypothesis $Q(\alpha) \notin \mathcal{T}$, thus (2) implies $\alpha \notin \mathsf{dom}(\widehat{Q})$, that is, $\widehat{Q}(\alpha) = \alpha$, which is the expected result. Conversely, we have by hypothesis $Q(\alpha) \in \mathcal{T}$, that is $\sigma \in \mathcal{T}$. Then $\widehat{Q}(\alpha)$ is $\widehat{Q_1}(\mathsf{nf}(\sigma))$ $(\underline{\mathbf{3}})$. By hypothesis (1), $\mathsf{nf}(\sigma)$ is not in ϑ, hence, neither is $\widehat{Q_1}(\mathsf{nf}(\sigma))$. By (3), this implies $\widehat{Q}(\alpha) \notin \vartheta$. This is the expected result.

Property v: By Property i, we have $(Q)\ Q[\alpha] \equiv Q[\beta]$ $(\underline{\mathbf{1}})$. If $Q[\alpha]$ is $Q[\beta]$, then $Q(\alpha) = Q(\beta)$ by definition, which gives the expected result by Eq-Refl. Otherwise, by Corollary 1.5.10 and (1), we have $\widehat{Q}(Q[\alpha]) = \widehat{Q}(Q[\beta])$. If $\widehat{Q}(Q[\alpha])$ is a type variable γ, then $Q(Q[\alpha]) = Q(\gamma)$ by Property ii, that is, $Q(\alpha) = Q(\gamma)$. Similarly, $Q(\beta) = Q(\gamma)$, thus $Q(\alpha) = Q(\beta)$ and the result holds by Eq-Refl. Otherwise, $\widehat{Q}(Q[\alpha]) \notin \vartheta$ and by Property iii, we have $(Q)\ Q[\alpha] \equiv Q(Q[\alpha])$, that is, $(Q)\ Q[\alpha] \equiv Q(\alpha)$ $(\underline{\mathbf{2}})$. Similarly, $(Q)\ Q[\beta] \equiv Q(\beta)$ $(\underline{\mathbf{3}})$. By (1), (2), and (3), we get $(Q)\ Q(\alpha) \equiv Q(\beta)$. This is the expected result.

Property vi: By hypothesis, $(Q)\ \alpha \Diamond \sigma$ $(\underline{\mathbf{1}})$ holds. By Lemma 2.1.6 on (1), we must have $(Q)\ \sigma \equiv \alpha$. By Lemma 1.5.9 and Property 1.5.6.iii (page 51), we have $\widehat{Q}(\mathsf{nf}(\sigma)) \approx \widehat{Q}(\alpha)$, that is $\widehat{Q}(\mathsf{nf}(\sigma)) = \widehat{Q}(\alpha)$ $(\underline{\mathbf{2}})$ since $\widehat{Q}(\alpha)$ is a monotype (does not have any quantifier). By hypothesis, $Q(\alpha) \notin \mathcal{T}$. Hence, by Property iv, we have $\widehat{Q}(\alpha) \in \vartheta$, that is, $\widehat{Q}(\alpha) \in \mathsf{dom}(Q)$ since Q is closed by well-formedness of (1). Hence, $\widehat{Q}(\mathsf{nf}(\sigma)) \in \mathsf{dom}(Q)$ holds from (2). Necessarily, $\mathsf{nf}(\sigma)$ is a type variable β such that $\widehat{Q}(\beta) \in \mathsf{dom}(Q)$. By Property iv, we get $Q(\beta) \notin \mathcal{T}$. ■

Proof of Lemma 2.3.1

We first prove the two following results, namely $(\underline{\mathbf{1}})$ and $(\underline{\mathbf{2}})$:

For any σ, there exists a restricted derivation of $\sigma \equiv \mathsf{nf}(\sigma)$.

If σ_a *and* σ_b *are in normal forms, and if* $\sigma_a \approx \sigma_b$ *holds, then there exists a restricted derivation of* $\sigma_a \equiv \sigma_b$*.*

Proof sketch of (1): Property 1.5.6.i states that $\sigma \equiv \mathsf{nf}(\sigma)$ holds. We have to show that this derivation is, moreover, restricted. Actually, the proof of Property 1.5.6.i (page 51) is still appropriate. Indeed, the proof of Property 1.5.6.i (page 51) does not use R-Context-L, and builds only restricted derivations. $\square$

Proof sketch of (2): The proof of $\sigma_a \equiv \sigma_b$ only uses rules R-Context-R, R-Context-L, and Eq-Comm. Every occurrence of Rule R-Context-L is necessarily restricted since σ_a and σ_b are in normal form: for instance, if σ_a is $\forall\,(\alpha \diamond \sigma_1)\ \sigma$, we must have $\alpha \in \mathsf{ftv}(\sigma)$, $\mathsf{nf}(\sigma) \neq \alpha$, and $\mathsf{nf}(\sigma) \neq \bot$. $\square$

We first prove the result for a derivation of $(Q)\ \sigma_1 \equiv \sigma_2$: By Lemma 1.5.9, we have $\widehat{Q}(\mathsf{nf}(\sigma_1)) \approx \widehat{Q}(\mathsf{nf}(\sigma_2))$. Hence, $\widehat{Q}(\mathsf{nf}(\sigma_1)) \equiv \widehat{Q}(\mathsf{nf}(\sigma_2))$ **(<u>3</u>)** holds and is restricted by Result (2). By Property 1.5.6.iii (page 51), (3) can also be written $\mathsf{nf}(\widehat{Q}(\sigma_1)) \equiv \mathsf{nf}(\widehat{Q}(\sigma_2))$ **(<u>4</u>)**. By Result (1), we have restricted derivations of $\widehat{Q}(\sigma_1) \equiv \mathsf{nf}(\widehat{Q}(\sigma_1))$ **(<u>5</u>)** and $\mathsf{nf}(\widehat{Q}(\sigma_2)) \equiv \widehat{Q}(\sigma_2)$ **(<u>6</u>)**. Besides, $(Q)\ \sigma_1 \equiv \widehat{Q}(\sigma_1)$ **(<u>7</u>)** holds by Eq-Mono and is restricted. Similarly, $(Q)\ \widehat{Q}(\sigma_2) \equiv \sigma_2$ **(<u>8</u>)** is restricted. By Property 1.5.3.v (page 49), R-Trans, (7), (5), (4), (6), and (8), we have a restricted derivation of $(Q)\ \sigma_1 \equiv \sigma_2$. This is the expected result.

In summary, the following rule is admissible:

$$\textsc{Equiv-R}\quad \frac{(Q)\ \sigma_1 \equiv \sigma_2}{(Q)\ \sigma_1 \diamondsuit \sigma_2 \text{ (restricted)}}$$

Then we prove the result for an abstraction or instantiation derivation. The proof is by induction on the derivation.

○ Case I-Bot: We proceed by case analysis.

Subcase $\mathsf{nf}(\sigma)$ is $\bot$: Then $(Q)\ \bot \equiv \sigma$ holds by Property 1.5.6.i (page 51). We conclude by Rule Equiv-R.

Subcase $\sigma \in \mathcal{V}$: Then $\mathsf{nf}(\sigma) = \alpha$ by definition. By Property 1.5.6.ii (page 51), we have $\alpha \in \mathsf{ftv}(\sigma)$. By well-formedness, we must have $\alpha \in \mathsf{dom}(Q)$. Let α' be $Q[\alpha]$. We have $(\alpha' \diamond' \sigma') \in Q$, $\sigma' \notin \mathcal{V}$ **(<u>9</u>)**, and $(Q)\ \alpha \equiv \alpha'$ **(<u>10</u>)** by Property 2.2.2.i (page 69). By Rule Equiv-R and (10), we have a restricted derivation of $(Q)\ \alpha \diamondsuit \alpha'$ **(<u>11</u>)**. Moreover, if $\mathsf{nf}(\sigma')$ is not $\bot$, we have a restricted derivation of $(Q)\ \bot \sqsubseteq \sigma'$ **(<u>12</u>)** by I-Nil and (9). If $\mathsf{nf}(\sigma')$ is $\bot$, $(Q)\ \bot \equiv \sigma'$ by Property 1.5.6.i (page 51), and (12) holds by Equiv-R. In both cases, we have a restricted derivation of (12). If σ' is in $\mathcal{T}$, then $(Q)\ \sigma' \sqsubseteq \alpha'$ **(<u>13</u>)** holds by Eq-Mono and I-Equiv*. Otherwise, (13) holds by I-Hyp. In both cases, (13) is restricted. From (12), (13), (11) and R-Trans, we get a restricted derivation of $(Q)\ \bot \sqsubseteq \alpha$ This is the expected result.

Otherwise, the judgment is already restricted.

◦ Case R-Context-Rigid and R-Context-Flexible: The premise is the judgment $(Q)\ \sigma_1 \diamondsuit \sigma_2$ **(<u>14</u>)** and the conclusion is $(Q)\ \forall(\alpha \diamond \sigma_1)\ \sigma \diamondsuit \forall(\alpha \diamond \sigma_2)\ \sigma$. By induction hypothesis, we have a restricted derivation of (14). If $\alpha \notin \mathsf{ftv}(\sigma)$, then $(Q)\ \forall(\alpha\diamond\sigma_1)\ \sigma \equiv \forall(\alpha\diamond\sigma_2)\ \sigma$ is derivable by Eq-Free and R-Trans, and we conclude by Equiv-R. If $\mathsf{nf}(\sigma)$ is α, then $(Q)\ \forall(\alpha \diamond \sigma_1)\ \sigma \equiv \sigma_1$ **(<u>15</u>)** holds by Property 1.5.6.i (page 51), R-Context-R, and Eq-Var. Similarly, $(Q)\ \forall(\alpha \diamond \sigma_2)\ \sigma \equiv \sigma_2$ **(<u>16</u>)** holds. By Equiv-R, (15), (16), and (14), we have a restricted derivation of $(Q)\ \forall(\alpha\diamond\sigma_1)\ \sigma \diamondsuit \forall(\alpha \diamond \sigma_2)\ \sigma$. This is the expected result. If $\mathsf{nf}(\sigma)$ is $\bot$, then $(Q)\ \forall(\alpha \diamond \sigma_1)\ \sigma \equiv \bot$ and $(Q)\ \forall(\alpha\diamond\sigma_2)\ \sigma \equiv \bot$ hold. By R-Trans and Equiv-R, we have a restricted derivation of $(Q)\ \forall(\alpha \diamond \sigma_1)\ \sigma \diamondsuit \forall(\alpha \diamond \sigma_2)\ \sigma$. Otherwise, the judgment is restricted, and there exists a restricted derivation of the premise (14). This is the expected result.

◦ Case R-Context-R: If $(Q, \alpha\diamond\sigma)\ \sigma_1 \equiv \sigma_2$ holds, then $(Q)\ \forall(\alpha\diamond\sigma)\ \sigma_1 \equiv \forall(\alpha\diamond\sigma)\ \sigma_2$ holds by R-Context-R, and we get the expected result by A-Equiv or I-Equiv*.

◦ Case I-Rigid: If $\alpha \notin \mathsf{ftv}(\sigma)$, then we can derive $(Q)\ \forall(\alpha \geq \sigma_1)\ \sigma \equiv \sigma$ by Eq-Free, as well as $(Q)\ \sigma \equiv \forall(\alpha = \sigma_1)\ \sigma$. We conclude by Equiv-R. If σ_1 is in $\mathcal{T}$, then $\mathsf{nf}(\sigma_1)$ is τ_1 by definition and we have $(Q)\ \forall(\alpha \geq \sigma_1)\ \sigma \equiv \sigma[\tau_1/\alpha]$ by Eq-Mono, as well as $(Q)\ \sigma[\tau_1/\alpha] \equiv \forall(\alpha = \sigma_1)\ \sigma$. We conclude by Equiv-R. If $\mathsf{nf}(\sigma)$ is α, then $(Q)\ \forall(\alpha \geq \sigma_1)\ \sigma \equiv \sigma_1$ by Property 1.5.6.i (page 51) and Eq-Var, as well as $(Q)\ \sigma_1 \equiv \forall(\alpha = \sigma_1)\ \sigma$. We conclude by Equiv-R. Otherwise, the judgment is already restricted.

◦ Case A-Hyp: We have $(\alpha_1 = \sigma_1) \in Q$. If $\sigma_1 \in \mathcal{T}$, then $(Q)\ \sigma_1 \equiv \tau_1$ **(<u>17</u>)** holds for some monotype τ_1 by Properties 1.5.11.ii (page 54) and 1.5.3.v (page 49). Hence, $(Q)\ \alpha_1 \equiv \tau_1$ **(<u>18</u>)** holds by Eq-Mono. Finally, $(Q)\ \alpha_1 \equiv \sigma_1$ by (18), (17), and R-Trans. By Rule Equiv-R, we get the expected result $(Q)\ \sigma_1 \sqsubseteq \alpha_1$.

◦ Case I-Hyp: similar.

◦ All other cases are by induction hypothesis. ■

Proof of Lemma 2.3.3

We first show the three following properties, then we show the lemma.

1. For any type σ, there exists a thrifty derivation of $\sigma \equiv \mathsf{nf}(\sigma)$.
2. Any equivalence derivation can be rewritten into a thrifty derivation.
3. Any derivation can be rewritten into a thrifty derivation.

<u>Property 1</u> : It is by induction on the number of universal quantifiers appearing in σ. If σ has no universal quantifiers, then σ is $\bot$ or τ, and we get the result by Rule Eq-Refl

since $\mathsf{nf}(\sigma)$ is σ itself. Otherwise, σ is $\forall\,(\alpha \diamond \sigma_1)\ \sigma_2$ ($\underline{\mathbf{1}}$), where σ_1 and σ_2 have strictly less universal quantifiers than σ. We proceed by case analysis:

∘ Case $\mathsf{nf}(\sigma_2)$ is α: Then $\mathsf{nf}(\sigma)$ is $\mathsf{nf}(\sigma_1)$, and $\sigma_2 \equiv \alpha$ holds and is thrifty by induction hypothesis. By Property 1.5.3.v (page 49), $(\alpha \diamond \sigma_1)\ \sigma_2 \equiv \alpha$ also holds, and is thrifty. Hence, $\forall\,(\alpha \diamond \sigma_1)\ \sigma_2 \equiv \forall\,(\alpha \diamond \sigma_1)\ \alpha$ ($\underline{\mathbf{2}}$) holds by R-Context-R. This equivalence is thrifty since $\mathsf{nf}(\sigma_2)$ and $\mathsf{nf}(\alpha)$ are both α. We have $\forall\,(\alpha \diamond \sigma_1)\ \alpha \equiv \sigma_1$ by Eq-Var ($\underline{\mathbf{3}}$). Since σ is $\forall\,(\alpha \diamond \sigma_1)\ \sigma_2$ (from (1)), we get $\sigma \equiv \sigma_1$ by R-Trans, (2) and (3). By induction hypothesis we also have a thrifty derivation of $\sigma_1 \equiv \mathsf{nf}(\sigma_1)$, therefore $\sigma \equiv \mathsf{nf}(\sigma_1)$ holds by R-Trans and is thrifty. In the following, we assume $\mathsf{nf}(\sigma_2)$ is not α.

∘ Case $\mathsf{nf}(\sigma_1)$ is τ_1 ($\underline{\mathbf{4}}$): We have a thrifty derivation of $\sigma_2 \equiv \mathsf{nf}(\sigma_2)$ by induction hypothesis, thus we get a thrifty derivation of $\sigma \equiv \forall\,(\alpha \diamond \sigma_1)\ \mathsf{nf}(\sigma_2)$ by R-Context-R. Moreover, we have $\sigma_1 \equiv \tau_1$ by induction hypothesis on (4), hence we have $\sigma \equiv \mathsf{nf}(\sigma_2)[\tau_1/\alpha]$ by Rule Eq-Mono* and R-Trans. This derivation is thrifty. We conclude by observing that $\mathsf{nf}(\sigma)$ is $\mathsf{nf}(\sigma_2)[\tau_1/\alpha]$.

∘ Case $\alpha \notin \mathsf{ftv}(\sigma_2)$: Then $\mathsf{nf}(\sigma)$ is $\mathsf{nf}(\sigma_2)$, and $\sigma \equiv \sigma_2$ by Eq-Free, thus we get a thrifty derivation of $\sigma \equiv \mathsf{nf}(\sigma_2)$ by induction hypothesis and R-Trans.

∘ Otherwise, $\mathsf{nf}(\sigma)$ is $\forall\,(\alpha \diamond \mathsf{nf}(\sigma_1))\ \mathsf{nf}(\sigma_2)$. By induction hypothesis, $\sigma_1 \equiv \mathsf{nf}(\sigma_1)$ and $\sigma_2 \equiv \mathsf{nf}(\sigma_2)$. Hence, $(\alpha \diamond \sigma_1)\ \sigma_2 \equiv \mathsf{nf}(\sigma_2)$ holds by Property 1.5.3.v (page 49). By rules R-Context-L, R-Context-R and R-Trans, we get a thrifty derivation of $\sigma \equiv \mathsf{nf}(\sigma)$.

Property 2: By hypothesis, we have $(Q)\ \sigma_1 \equiv \sigma_2$. By Corollary 1.5.10, we have a derivation of $\widehat{Q}(\sigma_1) \equiv \widehat{Q}(\sigma_2)$ ($\underline{\mathbf{1}}$). Note that $(Q)\ \sigma_1 \equiv \widehat{Q}(\sigma_1)$ ($\underline{\mathbf{2}}$) is thrifty since it only uses Eq-Mono. Similarly, $(Q)\ \widehat{Q}(\sigma_2) \equiv \sigma_2$ ($\underline{\mathbf{3}}$) is thrifty. By Property 1.5.11.i (page 54) applied to (1), we have $\mathsf{nf}(\widehat{Q}(\sigma_1)) \approx \mathsf{nf}(\widehat{Q}(\sigma_2))$ ($\underline{\mathbf{4}}$). The derivation of (4) provides a derivation of $\mathsf{nf}(\widehat{Q}(\sigma_1)) \equiv \mathsf{nf}(\widehat{Q}(\sigma_2))$ ($\underline{\mathbf{5}}$) by Property 1.5.3.iii (page 49). Besides, the derivation of (5) is thrifty: indeed, each occurrence or Rule R-Context-R operates on normal forms. By Property 1, we have a thrifty derivation of $\widehat{Q}(\sigma_1) \equiv \mathsf{nf}(\widehat{Q}(\sigma_1))$ ($\underline{\mathbf{6}}$) and $\mathsf{nf}(\widehat{Q}(\sigma_2)) \equiv \widehat{Q}(\sigma_2)$ ($\underline{\mathbf{7}}$). By R-Trans, (2), (6), (5), (7), and (3), we get a thrifty derivation of $(Q)\ \sigma_1 \equiv \sigma_2$.

Before going on proving the remaining properties, we prove the following result, which we refer to as Thrifty-Var.

> *Assume $(\alpha \diamond \sigma) \in Q$. If $(Q)\ \sigma_1 \mathrel{\Diamond} \sigma_2$ is thrifty, and $(Q)\ \sigma_2 \equiv \alpha$ holds but $(Q)\ \sigma_1 \equiv \alpha$ does not hold, then there exists a thrifty derivation of $(Q)\ \sigma_1 \mathrel{\Diamond} \sigma$. Besides, if $\Diamond$ is $\sqsubseteq$, then $\diamond$ is rigid.*

Proof: By induction on the derivation. None of the equivalence cases occur since σ_1 is not equivalent to α under Q, while σ_2 is.

∘ Case R-Trans The premises are $(Q)\ \sigma_1 \Diamond \sigma'_1$ $\mathbf{(\underline{1})}$ and $(Q)\ \sigma'_1 \Diamond \sigma_2$ $\mathbf{(\underline{2})}$. Besides, both derivations are thrifty. If $(Q)\ \sigma'_1 \equiv \alpha$, we conclude directly by induction hypothesis on (1). Otherwise, we know by induction hypothesis on (2) that we have a thrifty derivation of $(Q)\ \sigma'_1 \Diamond \sigma$ $\mathbf{(\underline{3})}$. Hence, by R-Trans, (1), and (3), we have a thrifty derivation of $(Q)\ \sigma_1 \Diamond \sigma$. Additionally, if $\Diamond$ is $\sqsubseteq$, then $\diamond$ is rigid.

∘ Case R-Context-R We have $\sigma_1 = \forall\,(\beta \diamond' \sigma')\ \sigma'_1$ $\mathbf{(\underline{4})}$ and $\sigma_2 = \forall\,(\beta \diamond' \sigma')\ \sigma'_2$. The premise is $(Q, \beta \diamond' \sigma')\ \sigma'_1 \Diamond \sigma'_2$ $\mathbf{(\underline{5})}$ and is thrifty. By hypothesis, we have $(Q)\ \forall\,(\beta \diamond' \sigma')\ \sigma'_2 \equiv \alpha$ $\mathbf{(\underline{6})}$.

Subcase $\mathsf{nf}(\sigma'_2)$ is β: By Eq-Var, R-Trans, and (6), we get $(Q)\ \sigma' \equiv \alpha$ $\mathbf{(\underline{7})}$. Moreover, $\mathsf{nf}(\sigma'_1)$ is not β (otherwise, we would have $(Q)\ \sigma_1 \equiv \alpha$ by Property 1.5.6.i (page 51), (7), (4), and Eq-Var). Hence, this occurrence of R-Context-R is squandering, which is not possible by hypothesis.

Subcase $\beta \notin \mathsf{ftv}(\sigma'_2)$: We have $\mathsf{nf}(\sigma) \neq \beta$ $\mathbf{(\underline{8})}$ by Property 1.5.6.ii (page 51). Since by hypothesis this occurrence of R-Context-R or R-Context-R is thrifty, we must have $\mathsf{nf}(\sigma'_1) \neq \beta$ $\mathbf{(\underline{9})}$. By Eq-Free, we have $(Q)\ \sigma'_2 \equiv \alpha$. By induction hypothesis on (5), we have a thrifty derivation of $(Q, \beta \diamond' \sigma')\ \sigma'_1 \Diamond \sigma$ $\mathbf{(\underline{10})}$. Besides, if $\Diamond$ is $\sqsubseteq$, then $\diamond$ is rigid. By (10), R-Context-R, we get $(Q)\ \forall\,(\beta \diamond' \sigma')\ \sigma'_1 \Diamond \forall\,(\beta \diamond' \sigma')\ \sigma$ $\mathbf{(\underline{11})}$. This derivation is thrifty since $\mathsf{nf}(\sigma'_1) \neq \beta$ (from (9)) and $\mathsf{nf}(\sigma) \neq \beta$ (from (8)). By Eq-Free, we have $(Q)\ \forall\,(\alpha \diamond' \sigma')\ \sigma \equiv \sigma$. Hence, $(Q)\ \forall\,(\alpha \diamond' \sigma')\ \sigma \Diamond \sigma$ $\mathbf{(\underline{12})}$ holds by A-Equiv and I-Abstract, and is thrifty. By R-Trans, (11), and (12), we have a thrifty derivation of $(Q)\ \forall\,(\beta \diamond' \sigma')\ \sigma'_1 \Diamond \sigma$. This is the expected result.

Otherwise $\beta \in \mathsf{ftv}(\sigma'_2)$ $\mathbf{(\underline{13})}$ and $\mathsf{nf}(\sigma'_2)$ is not β $\mathbf{(\underline{14})}$. We show that $\sigma \in \mathcal{T}$. Let θ be $\widehat{Q}$. note that $\beta \notin \mathsf{dom}(\theta) \cup \mathsf{codom}(\theta)$ by well-formedness of (5). By Corollary 1.5.10 applied to (6), we have $\theta(\forall\,(\beta \diamond' \sigma')\ \sigma'_2) \equiv \theta(\alpha)$ $\mathbf{(\underline{15})}$. By (13), there exists an occurrence u such that $\sigma'_2/u = \beta$ $\mathbf{(\underline{16})}$. We cannot have $\sigma'_2 \equiv \beta$, because of (14) and Property 1.5.11.i (page 54). By Property 2.1.5.ii (page 67), this implies $\sigma'_2/\epsilon \neq \beta$. Hence, u is not ϵ in (16). As a consequence, σ'_2/ϵ must be a type constructor g. By Property 1.5.4.i (page 50) applied to (15), we have $\theta(\alpha)/\epsilon = g$. Hence, α must be in $\mathsf{dom}(\theta)$, which means that $\sigma \in \mathcal{T}$ $\mathbf{(\underline{17})}$. By Property 1.5.11.ii (page 54), there exists τ such that $\sigma \equiv \tau$. In summary, $(\alpha \diamond \sigma) \in Q$, and $\sigma \equiv \tau$ $\mathbf{(\underline{18})}$ holds. We have $(Q)\ \alpha \equiv \tau$ $\mathbf{(\underline{19})}$ by Eq-Mono, thus $(Q)\ \alpha \equiv \sigma$ $\mathbf{(\underline{20})}$ holds by R-Trans (18), and (19). By (6), (20), and R-Trans, we get $(Q)\ \sigma_2 \equiv \sigma$. By Property 2, we have a thrifty derivation of $(Q)\ \sigma_2 \equiv \sigma$ $\mathbf{(\underline{21})}$. By hypothesis, $(Q)\ \sigma_1 \Diamond \sigma_2$ $\mathbf{(\underline{22})}$ is thrifty, hence $(Q)\ \sigma_1 \Diamond \sigma$ is thrifty by (22), (21), and R-Trans. Besides since $\sigma \in \mathcal{T}$ (from (17)), the binding $(\alpha \diamond \sigma)$ is considered flexible and rigid. This is the expected result.

∘ Case A-Hyp: We have $(\alpha_1 = \sigma_1) \in Q$, and σ_2 is α_1. By hypothesis, $(Q)\ \alpha_1 \equiv \alpha$ $\mathbf{(\underline{23})}$.

Subcase α is α_1: We have $\sigma_1 = \sigma$ and $(Q)\ \sigma_1 \sqsubseteq \sigma$ holds by Eq-Refl, and is thrifty. Besides, $\Diamond$ is $\sqsubseteq$ and $\diamond$ is rigid.

Otherwise, σ_1 cannot be in $\mathcal{T}$ (otherwise we would have $(Q)\ \sigma_1 \equiv \alpha$ by (23) and Eq-Mono). By Property 1.5.11.vii (page 54) and (23), we have $\widehat{Q}(\alpha_1) = \widehat{Q}(\alpha)$, that is, $\alpha_1 = \widehat{Q}(\alpha)$. As a consequence, we must have $\alpha \in \mathsf{dom}(\widehat{Q})$, that is, $\sigma \in \mathcal{T}$. Then $(Q)\ \alpha \equiv \sigma$ $\mathbf{(\underline{24})}$ holds, thus $(Q)\ \alpha_1 \equiv \sigma$ $\mathbf{(\underline{25})}$ holds by R-Trans, (23), and (24). By Property 2, (25) is made thrifty. Hence, we get a thrifty derivation of $(Q)\ \alpha_1 \mathrel{-\!\!\!\!\sqsubseteq} \sigma$ $\mathbf{(\underline{26})}$ by A-Equiv. Additionally, $(Q)\ \sigma_1 \mathrel{-\!\!\!\!\sqsubseteq} \alpha_1$ $\mathbf{(\underline{27})}$ holds by A-Hyp and is thrifty by definition. Therefore, we get a thrifty derivation of $(Q)\ \sigma_1 \mathrel{-\!\!\!\!\sqsubseteq} \sigma$ by R-Trans, (27), and (26).

- Case I-Hyp is similar.
- Case I-Bot: Since σ_1 is $\bot$, we can derive $(Q)\ \sigma_1 \sqsubseteq \sigma$ by I-Bot, and it is thrifty.
- Case I-Abstract: By induction hypothesis.
- Case I-Rigid: We have $\sigma_1 = \forall\,(\alpha \geq \sigma'_1)\ \sigma'_2$ and $\sigma_2 = \forall\,(\alpha = \sigma'_1)\ \sigma'_2$. If $\alpha \notin \mathsf{ftv}(\sigma'_2)$, then $\sigma_1 \equiv \sigma_2$, which is a contradiction. Since $(Q)\ \sigma_2 \equiv \alpha$ holds by hypothesis, we must have $\sigma_2 \in \mathcal{T}$ by Property 1.5.11.x (page 54). Hence, $\sigma'_2 \in \mathcal{T}$ and $\sigma'_1 \in \mathcal{T}$ by Property 2.1.5.i (page 67). Then $\sigma_1 \equiv \sigma_2$ by Eq-Mono and R-Trans, which is a contradiction. Hence, this case cannot occur.
- Case R-Context-Rigid and R-Context-Flexible: We have $\sigma_1 = \forall\,(\beta \diamond \sigma'_1)\ \sigma_0$ and $\sigma_2 = \forall\,(\beta \diamond \sigma'_2)\ \sigma_0$. The premise is $(Q)\ \sigma'_1 \diamondsuit \sigma'_2$ $\mathbf{(\underline{28})}$, and the conclusion is $(Q)\ \sigma_1 \diamondsuit \sigma_2$ $\mathbf{(\underline{29})}$. Moreover, by hypothesis, we have $(Q)\ \sigma_2 \equiv \alpha$ $\mathbf{(\underline{30})}$, that is, $(Q)\ \forall\,(\beta \diamond \sigma'_2)\ \sigma_0 \equiv \alpha$ $\mathbf{(\underline{31})}$ holds.

Subcase $\mathsf{nf}(\sigma_0)$ is β: Then $(Q)\ \sigma'_2 \equiv \alpha$ $\mathbf{(\underline{32})}$ holds by Eq-Var and (31). Besides, $(Q)\ \sigma'_1 \equiv \alpha$ does not hold $\mathbf{(\underline{33})}$ otherwise, we would have $(Q)\ \sigma_1 \equiv \alpha$. By induction hypothesis on (28), (32), and (33), we have a thrifty derivation of $(Q)\ \sigma'_1 \diamondsuit \sigma$, which gives $(Q)\ \forall\,(\beta \diamond \sigma'_1)\ \sigma_0 \diamondsuit \forall\,(\beta \diamond \sigma)\ \sigma_0$ by R-Context-Rigid or R-Context-Flexible. Then by Eq-Var, we get $(Q)\ \forall\,(\beta \diamond \sigma'_1)\ \sigma_0 \diamondsuit \sigma$. Additionally, by induction hypothesis on (28), if $\diamondsuit$ is $\mathrel{-\!\!\!\!\sqsubseteq}$, then $\diamond$ is rigid.

Subcase $\beta \notin \mathsf{ftv}(\sigma_0)$: Then $(Q)\ \sigma_1 \equiv \sigma_2$ holds by Eq-Free, which is a contradiction.

Otherwise, $\beta \in \mathsf{ftv}(\sigma_0)$ and $\mathsf{nf}(\sigma_0)$ is not β. Then σ_0/ϵ must be a type constructor g. By Property 1.5.11.vi (page 54) and (31), we must have $\widehat{Q}(\alpha)/\epsilon = g$, which implies $\alpha \in \mathsf{dom}(\widehat{Q})$, that is $\sigma \in \mathcal{T}$ $\mathbf{(\underline{34})}$. As a consequence, $(Q)\ \alpha \equiv \sigma$ $\mathbf{(\underline{35})}$ holds, thus $(Q)\ \sigma_2 \equiv \sigma$ $\mathbf{(\underline{36})}$ holds by R-Trans, (30), and (35). By Property 2 and (36), there exists a thrifty derivation of $(Q)\ \sigma_2 \equiv \sigma$. By A-Equiv or I-Equiv*, $(Q)\ \sigma_2 \diamondsuit \sigma$ $\mathbf{(\underline{37})}$ holds and is thrifty. Then $(Q)\ \sigma_1 \diamondsuit \sigma$ holds by R-Trans or R-Trans, (29) and (37), and is thrifty. Since $\sigma \in \mathcal{T}$ (from (34)), the binding $(\alpha \diamond \sigma)$ is flexible and rigid. ☐

Thanks to Thrifty-Var, we can prove remaining properties:
Property 3: We show that any derivation with exactly one squandering rule, used last, can be made thrifty and keep the same conclusion. The result for any given derivation

is immediate by induction on the number of squandering rule of the derivation. By definition, there are two ways of being a squandering rule:

- First case: The conclusion is $(Q)\ \forall\,(\alpha\diamond\sigma)\ \sigma_1 \mathrel{\Diamond} \forall\,(\alpha\diamond\sigma)\ \sigma_2$, where $\mathsf{nf}(\sigma_2)$ is α and $\mathsf{nf}(\sigma_1)$ is not α. The last rule used is X-Context-R, thus the premise is a derivation of $(Q, \alpha\diamond\sigma)\ \sigma_1 \mathrel{\Diamond} \sigma_2$ $\mathbf{(\underline{1})}$, which is thrifty by hypothesis. By Property 1.5.6.i (page 51), $\sigma_2 \equiv \alpha$ $\mathbf{(\underline{2})}$ holds.

  Subcase σ_1 is not equivalent to α under $(Q, \alpha\diamond\sigma)$: By thrifty-var and (1), there exists a thrifty derivation of $(Q, \alpha\diamond\sigma)\ \sigma_1 \mathrel{\Diamond} \sigma$. Hence, by X-Context-R, we get a thrifty derivation of $(Q)\ \forall\,(\alpha\diamond\sigma)\ \sigma_1 \mathrel{\Diamond} \forall\,(\alpha\diamond\sigma)\ \sigma$ $\mathbf{(\underline{3})}$. Additionally, $(Q)\ \forall\,(\alpha\diamond\sigma)\ \sigma \equiv \sigma$ $\mathbf{(\underline{4})}$ holds by Eq-Free, and $(Q)\ \sigma \equiv \forall\,(\alpha\diamond\sigma)\ \alpha$ $\mathbf{(\underline{5})}$ by Eq-Var. Finally, by R-Trans, (4), and (5), and (2), we get $(Q)\ \forall\,(\alpha\diamond\sigma)\ \sigma \equiv \forall\,(\alpha\diamond\sigma)\ \sigma_2$ $\mathbf{(\underline{6})}$. By Property 2, this equivalence can be made thrifty. Hence, by R-Trans, (3), and (6), we have a thrifty derivation of $(Q)\ \forall\,(\alpha\diamond\sigma)\ \sigma_1 \mathrel{\Diamond} \forall\,(\alpha\diamond\sigma)\ \sigma_2$. This is the expected result.

  Subcase σ_1 is equivalent to α under $(Q, \alpha\diamond\sigma)$: By Lemma 2.1.6 and (1), we get $(Q, \alpha\diamond\sigma)\ \sigma_2 \equiv \alpha$. Hence, $(Q)\ \forall\,(\alpha\diamond\sigma)\ \sigma_1 \equiv \forall\,(\alpha\diamond\sigma)\ \sigma_2$ holds by R-Context-R. By Property 2, we have a thrifty derivation of $(Q)\ \forall\,(\alpha\diamond\sigma)\ \sigma_1 \mathrel{\Diamond} \forall\,(\alpha\diamond\sigma)\ \sigma_2$.
- Second case: The conclusion is $(Q)\ \forall\,(\alpha\diamond\sigma)\ \sigma_1 \mathrel{\Diamond} \forall\,(\alpha\diamond\sigma)\ \sigma_2$, where $\mathsf{nf}(\sigma_2)$ is not α and $\mathsf{nf}(\sigma_1)$ is α. By Property 1.5.6.i (page 51), we have $\sigma_1 \equiv \alpha$. The premise is a thrifty derivation of $(Q, \alpha\diamond\sigma)\ \sigma_1 \mathrel{\Diamond} \sigma_2$. By Lemma 2.1.6, we have $(Q, \alpha\diamond\sigma)\ \sigma_1 \equiv \sigma_2$. Hence, $(Q)\ \forall\,(\alpha\diamond\sigma)\ \sigma_1 \equiv \forall\,(\alpha\diamond\sigma)\ \sigma_2$ holds by R-Context-R, and we conclude directly by Property 2.

Lemma 2.3.3 (proof sketch): We use Property 3 and we prove that any thrifty derivation is kept thrifty when using the rewriting rules suggested in the proof of Lemma 2.3.1. As a matter of fact, none of them uses a right-context rule, thus no squandering rule can be introduced. This means that the derivation is kept thrifty. ■

Proof of Corollary 2.3.4

By Property 3, we have a thrifty derivation of $(Q)\ \sigma_1 \mathrel{\Diamond} \alpha$ $\mathbf{(\underline{1})}$. If $(Q)\ \sigma_1 \equiv \alpha$ does not hold, then we get the expected result by thrifty-var (to be found in the preceding proof) and (1). Otherwise, we have $(Q)\ \sigma_1 \equiv \alpha$ $\mathbf{(\underline{2})}$ and $\sigma_1 \notin \mathcal{V}$. This implies $\alpha \in \mathsf{dom}(\widehat{Q})$ by Corollary 1.5.10 and 1.5.4.i (page 50). Hence, $\sigma \in \mathcal{T}$ $\mathbf{(\underline{3})}$. Thus, $(Q)\ \alpha \equiv \sigma$ $\mathbf{(\underline{4})}$ holds and $(Q)\ \sigma_1 \equiv \sigma$ follows by R-Trans, (2), and (4). Since $\sigma \in \mathcal{T}$ (from (3)), the binding $(\alpha\diamond\sigma)$ is flexible and rigid. This is the expected result. ■

Proof of Property 2.4.3

Property i : It is proved by structural induction on the context.

Property ii: It is a consequence of Properties i and 2.1.5.i.

Property iii : It is by structural induction on C. The cases $[\,]$, and $\forall\,(\alpha \diamond C)\ \sigma$ when $\alpha \notin \mathsf{ftv}(\sigma)$ are immediate. The cases $\mathsf{level}(C) = 0$, $\forall\,(\alpha \diamond \sigma)\ C'$, and $\forall\,(\alpha \diamond C')\ \sigma_a$ with $\sigma_a \equiv \alpha$ are by induction hypothesis. The last case is $C = \forall\,(\alpha \diamond C')\ \sigma$, when $\alpha \in \mathsf{ftv}(\sigma)$ $\mathbf{(\underline{1})}$, $\sigma \notin \mathcal{V}$ and $\mathsf{level}(C') \neq 0$ $\mathbf{(\underline{2})}$. By definition of (1), there exists an occurrence u such that $\sigma/u = \alpha$. By induction hypothesis and (2), C' is useful. Let β and γ be two fresh variables. By Property i, there exists u' such that $C'(\beta)/u' = \beta$ and $C'(\gamma)/u' = \gamma$. Hence, we have $C(\beta)/uu' = \beta$ and $C(\gamma)/uu' = \gamma$. By corollary 1.5.11.vi (page 54), we cannot have $C(\beta) \equiv C(\gamma)$. Hence, C is useful. ■

Proof of Property 2.4.4

Property i: We prove by structural induction that for any narrow context C_n, we have $\mathsf{level}(C_n) = 1$ (straightforward). Conversely, we assume that $\mathsf{level}(C) = 1$ $\mathbf{(\underline{1})}$ We prove by structural induction on C that C is narrow. If C is $[\,]$, then C is narrow. If C is $\forall\,(\alpha \diamond \sigma)\ C'$ $\mathbf{(\underline{2})}$, then $\mathsf{level}(C') = \mathsf{level}(C)$ by definition of levels, thus $\mathsf{level}(C') = 1$ by (1). Hence, C' is narrow by induction hypothesis, which implies that C is narrow by (2). If C is $\forall\,(\alpha \diamond C')\ \sigma_\alpha$, where $\sigma_\alpha \equiv \alpha$, then we have $\mathsf{level}(C') = \mathsf{level}(C)$ by definition, thus $\mathsf{level}(C') = 1$ by (1). Hence, C' is narrow by induction hypothesis, which implies that C is narrow too. If C is $\forall\,(\alpha \diamond C')\ \sigma$ and $\sigma \not\equiv \alpha$, then, either $\mathsf{level}(C') = 0$, which implies $\mathsf{level}(C) = 0$ and this is a contradiction, or $\mathsf{level}(C') > 0$ and $\mathsf{level}(C) = \mathsf{level}(C') + 1$, thus $\mathsf{level}(C) > 1$, which is a contradiction too. This case is not possible.

Property ii: If $\mathsf{level}(C) = 1$, we know that C is narrow. We prove by structural induction on C that C is equivalent to $\forall\,(\overline{C})\ [\,]$.

∘ Case $[\,]$: immediate.

∘ Case $\forall\,(\alpha \diamond \sigma)\ C'$: By induction hypothesis C' is equivalent to $\forall\,(\overline{C'})\ [\,]$. Hence, C is equivalent to $\forall\,(\alpha \diamond \sigma)\ \forall\,(\overline{C'})\ [\,]$, which is the expected result since $\overline{\forall\,(\alpha \diamond \sigma)\ C'}$ is $(\alpha \diamond \sigma)\overline{C'}$.

∘ Case $\forall\,(\alpha \diamond C')\ \sigma_\alpha$: We have C equivalent to C', and we get the expected result by induction hypothesis and by observing that $\overline{C}$ is $\overline{C'}$.

Property iii: It is shown by induction on C:

∘ Case $[\,]$ cannot occur since $\mathsf{level}(C) > 1$.

∘ Case $\forall\,(\alpha \diamond C_1)\ \sigma_\alpha$ and $\sigma_\alpha \equiv \alpha$: Then $\mathsf{level}(C_1) > 1$ and C is equivalent to C_1, thus the result is by induction hypothesis.

∘ Case $\forall\,(\alpha \diamond C_1)\ \sigma$ and $\sigma \not\equiv \alpha$: Then $\mathsf{level}(C_1) = \mathsf{level}(C) - 1$. If $\mathsf{level}(C_1) = 1$, then we get the expected result by Property ii, taking $Q_1 = \overline{C_1}$. Otherwise, by induction hypothesis, C_1 is equivalent to $C'_1(\forall\,(\beta \diamond \forall\,(Q_1)\;[\;])\;\sigma')$, and we get the expected result by taking $C' = \forall\,(\alpha \diamond C'_1)\;\sigma$.

∘ Case $\forall\,(\alpha{\diamond}\sigma)\;C_1$: We have $\mathsf{level}(C) = \mathsf{level}(C_1) > 1$. We get the result by induction hypothesis.

Property iv: If $\mathsf{level}(C) = 1$, then we get the result by taking $C' = [\;]$ and $C_n = C$. Hence, we assume $\mathsf{level}(C) > 1$, and we prove the result by structural induction on C:

∘ Case $[\;]$: Not possible since $\mathsf{level}(C) > 1$.

∘ Case $\forall\,(\alpha \diamond \sigma)\;C'$: By induction hypothesis.

∘ Case $\forall\,(\alpha{\diamond}C')\;\sigma_\alpha$ and $\sigma_\alpha \equiv \alpha$: We have $\mathsf{level}(C') = \mathsf{level}(C)$, $\mathsf{dom}_1(C') = \mathsf{dom}_1(C)$, and C' is equivalent to C. Hence, we get the result by induction hypothesis.

∘ Case $\forall\,(\alpha{\diamond}C')\;\sigma$ and $\sigma \not\equiv \alpha$: We have $\mathsf{level}(C') = \mathsf{level}(C) - 1$. If $\mathsf{level}(C') = 1$, we get the expected result. Otherwise, we get the result by induction hypothesis.

Property v: It is shown by structural induction on C:

∘ Case $[\;]$: immediate.

∘ Case $\forall\,(\alpha \diamond \sigma')\;C'$: Since $C(\sigma) \in \mathcal{V}$, we must have $C'(\sigma) \in \mathcal{V}$, and we get the result by induction hypothesis.

∘ Case $\forall\,(\alpha{\diamond}C')\;\sigma_\alpha$ and $\sigma_\alpha \equiv \alpha$: Since $C(\sigma) \equiv C'(\sigma)$ by Eq-Var, we have $C'(\sigma) \in \mathcal{V}$, and we get the result by induction hypothesis, observing that $\mathsf{level}(C) = \mathsf{level}(C')$.

∘ Case $\forall\,(\alpha \diamond C')\;\sigma'$ and $\sigma' \not\equiv \alpha$: If $\mathsf{level}(C') = 0$, or $\alpha \notin \mathsf{ftv}(\sigma')$, then $\mathsf{level}(C) = 0$, which is the expected result. Otherwise, $\alpha \in \mathsf{ftv}(\sigma')$ and $\mathsf{level}(C') > 0$. Hence, $\mathsf{level}(C) = \mathsf{level}(C') + 1$ by definition. Thus we have $\mathsf{level}(C) > 1$. We must show that this case is not possible, that is, $C(\sigma)$ cannot be in $\mathcal{V}$. Indeed, we have $\alpha \in \mathsf{ftv}(\sigma')$ and $\sigma' \not\equiv \alpha$, thus $\sigma'/\epsilon \notin \vartheta$. Hence, $\forall\,(\alpha \diamond C'(\sigma))\;\sigma'/\epsilon \notin \vartheta$, which implies that $C(\sigma) \notin \mathcal{V}$ by Property 1.5.4.i (page 50). ■

Proof of Lemma 2.5.5

The relations $\sqsubseteq^{\bar\alpha}$ and $\sqsubseteq^{\bar\alpha}$ are included (respectively) in $\sqsubseteq$ and $\sqsubseteq$ for any set $\bar\alpha$. Indeed, the new set of rules defining (respectively) $\sqsubseteq^{\bar\alpha}$ and $\sqsubseteq^{\bar\alpha}$ is derivable from the old set of rules defining (respectively) $\sqsubseteq$ and $\sqsubseteq$.

Conversely, we have to show that $\sqsubseteq$ is included in $\sqsubseteq^{\emptyset}$. By Lemma 2.5.2 (page 78), this amounts to showing that $(\equiv|\sqsubseteq_C)^*$ is included in $\sqsubseteq^{\emptyset}$. Since $\sqsubseteq^{\emptyset}$ is transitive, it suffices to show that $(\equiv|\sqsubseteq_C)$ is included in $\sqsubseteq^{\emptyset}$. Additionally, $\equiv$ is included in $\sqsubseteq^{\emptyset}$ by RuleA-Equiv', thus we only need to prove the following property:

If $(Q)\;\sigma_1 \sqsubseteq_C \sigma_2$ **(1)** *holds, then* $(Q)\;\sigma_1 \sqsubseteq^{\emptyset} \sigma_2$ *holds too.*

Necessarily, Ac-Hyp is used to derive (1). Hence, we have $\sigma_1 = C_r(\sigma_0)$, $\sigma_2 = C_r(\alpha)$, and $(\alpha = \sigma_0) \in Q\overline{C_r}$. If $\alpha \in \mathsf{dom}(Q)$, then α is frozen in the derivation of $(Q)\ \sigma_1 \sqsubseteq \sigma_2$, that is, α is not introduced in $\bar{\alpha}$ by context rules. Hence, $(Q)\ \sigma_1 \sqsubseteq^{\bar{\alpha}} \sigma_2$ holds, by A-Hyp' and context rules. Otherwise, $\alpha \in \mathsf{dom}(\overline{C_r})$, which implies that the context C_r is of the form $C_a(\forall\,(\alpha = \sigma_0)\ C_b)$. In summary, we have:

$$\sigma_1 = C_a(\forall\,(\alpha = \sigma_0)\ C_b(\sigma_0))\ (\underline{\mathbf{2}}) \qquad \sigma_2 = C_a(\forall\,(\alpha = \sigma_0)\ C_b(\alpha)) \qquad (Q)\ \sigma_1 \sqsubseteq_C \sigma_2\ (\underline{\mathbf{3}})$$

We have to show that $(Q)\ \sigma_1 \sqsubseteq^{\emptyset} \sigma_2$ holds too. In fact, it suffices to show that $(Q\overline{C_a})$ $\forall\,(\alpha = \sigma_0)\ C_b(\sigma_0) \sqsubseteq^{\bar{\alpha}} (\mathsf{dom}(C_a))\forall\,(\alpha = \sigma_0)\ C_b(\alpha)$ $(\underline{\mathbf{4}})$ holds. Then context rules A-Context-L' and R-Context-R give the expected result. We show (4) by case on the level of C_b.

∘ Case $\mathsf{level}(C_b) = 1$: Then C_b is equivalent to $\forall\,(\overline{C_b})\ [\,]$ by Property 2.4.4.ii (page 74). Hence, $C_b(\sigma_0) \equiv \forall\,(\overline{C_b})\ \sigma_0$ $(\underline{\mathbf{5}})$ holds. Since (2) and (3) are well-formed, we must have $\mathsf{ftv}(\sigma_0) \subseteq \mathsf{dom}(Q\overline{C_a})$, and $\mathsf{ftv}(\sigma_0)\ \#\ \mathsf{dom}(\overline{C_b})$. Hence, $\forall\,(\overline{C_b})\ \sigma_0$ is equivalent to σ_0 $(\underline{\mathbf{6}})$ by Eq-Free. Similarly, we have $\forall\,(\alpha = \sigma_0)\ C_b(\alpha) \equiv \forall\,(\alpha = \sigma_0)\ \alpha$ $(\underline{\mathbf{7}})$ by Eq-Free, and $\forall\,(\alpha = \sigma_0)\ \alpha \equiv \sigma_0$ $(\underline{\mathbf{8}})$ by Eq-var. Hence, R-Trans, (7), and (8) give $\forall\,(\alpha = \sigma_0)\ C_b(\alpha) \equiv \sigma_0$ $(\underline{\mathbf{9}})$. Consequently, by (5), (6), and (9), we have $(Q\overline{C_a})\ \forall\,(\alpha = \sigma_0)\ C_b(\sigma_0) \equiv \forall\,(\alpha = \sigma_0)\ C_b(\alpha)$, which implies the expected result (4) by A-Equiv'.

∘ Case $\mathsf{level}(C_b) = 2$: Then by Property 2.4.4.iii (page 74), C_b is equivalent to $\forall\,(Q_1)$ $\forall\,(\alpha' = \forall\,(Q_2)\ [\,])\ \sigma'$, for some prefixes Q_1, Q_2. As above, we note that $\mathsf{ftv}(\sigma_0)\ \#$ $\mathsf{dom}(Q_1 Q_2)$, thus $\forall\,(\alpha = \sigma_0)\ C_b(\sigma_0) \equiv \forall\,(\alpha = \sigma_0)\ \forall\,(\alpha' = \sigma_0)\ \forall\,(Q_1)\ \sigma'$ $(\underline{\mathbf{10}})$ holds by Eq-Free and Eq-Comm, and $\forall\,(\alpha = \sigma_0)\ C_b(\alpha) \equiv \forall\,(\alpha = \sigma_0)\ \forall\,(\alpha' = \alpha)\ \forall\,(Q_1)\ \sigma'$ $(\underline{\mathbf{11}})$. Hence, by rules A-Alias', A-Equiv', (10), and (11), we derive $(Q\overline{C_a})\ \forall\,(\alpha = \sigma_0)$ $C_b(\sigma_0) \sqsubseteq^{\mathsf{dom}(C_a)} \forall\,(\alpha = \sigma_0)\ C_b(\alpha)$. This is the expected result (4).

∘ Case $\mathsf{level}(C_b) > 1$ $(\underline{\mathbf{12}})$: We prove the following result:

> *Given $n > 0$, for any rigid context C_r such that $\mathsf{level}(C_r) = n + 1$, and σ_0 such that $\mathsf{ftv}(\sigma_0) \cup \{\alpha\}\ \#\ \mathsf{dom}(\overline{C_r})$, there exists a rigid context C'_r such that $\mathsf{level}(C'_r) = n$, $C_r(\sigma_0) \sqsubseteq^{\emptyset} C'_r(\sigma_0)$ and $C_r(\alpha) \equiv C'_r(\alpha)$.*

Proof: This is proved by structural induction on C_r.

Subcase $[\,]$ is not possible since $\mathsf{level}(C_r) > 1$.

Subcase $\forall\,(\beta \diamond \sigma)\ C_1$: By induction hypothesis.

Subcase $\forall\,(\beta \diamond C_1)\ \sigma$ and $\sigma \equiv \beta$: Then C_r is equivalent to C_1, and we get the result by induction hypothesis.

Subcase $\forall\,(\beta = C_1)\ \sigma$ and $\sigma \not\equiv \beta$: If $\mathsf{level}(C_1) > 1$, then we get the result by induction hypothesis. Otherwise, $\mathsf{level}(C_1) = 2$, thus, by Properties 2.4.4.iii (page 74) and 2.4.4.ii (page 74), C_1 is equivalent to $\forall\,(Q'_1)\ \forall\,(\beta' = \forall\,(Q'_2)\ [\,])\ \sigma_1$ for some prefixes Q'_1 and Q'_2. Besides, we have $\mathsf{ftv}(\sigma_0)\ \#\ \mathsf{dom}(Q'_1 Q'_2)$ by hypothesis. This implies $C_1(\sigma_0) \equiv$

$\forall\,(\beta' = \sigma_0)\ \forall\,(Q_1)\ \sigma_1$ by Eq-Free and Eq-Comm. Hence, $C(\sigma_0) \equiv \forall\,(\beta = \forall\,(\beta' = \sigma_0)\ \forall\,(Q_1)\ \sigma_1)\ \sigma$. By A-Up', we get $C(\sigma_0) \mathrel{-\!\!\!\!\sqsubseteq} [\emptyset]\forall\,(\beta' = \sigma_0)\ \forall\,(\beta = \forall\,(Q_1)\ \sigma_1)\ \sigma$. Similarly, $C(\alpha) \equiv \forall\,(\beta' = \alpha)\ \forall\,(\beta = \forall\,(Q_1)\ \sigma_1)\ \sigma$. Hence, the context $C'_r = \forall\,(\beta' = [\ ])\ \forall\,(\beta = \sigma_1)\ \sigma$ is of level $n = 1$ and is appropriate. □

Back to the main proof. By well-formedness of (2), we have $\mathsf{ftv}(\sigma) \cup \{\alpha\} \# \mathsf{dom}(C_b)$. By (12), $\mathsf{level}(C_b) = n > 1$, hence there exists a rigid context C'_b of level $n-1$ such that $C_b(\sigma_0) \mathrel{-\!\!\!\!\sqsubseteq}^{\emptyset} C'_b(\sigma_0)$ and $C_b(\alpha) \equiv C'_b(\alpha)$. By immediate iteration, we build a sequence C^i_b of rigid contexts, such that $C_b(\sigma_0) \mathrel{-\!\!\!\!\sqsubseteq}^{\emptyset} C^n_b[\sigma_0]$ $\mathbf{(\underline{13})}$ holds by R-Trans, $C_b(\alpha) \equiv C^n_b[\alpha]$ $\mathbf{(\underline{14})}$ hold, and C^n_b is of level 1. Moreover, $\forall\,(\alpha = \sigma_0)\ C^n_b[\sigma_0] \mathrel{-\!\!\!\!\sqsubseteq}^{\bar{\alpha}} \forall\,(\alpha = \sigma_0)\ C^n_b[\alpha]$ $\mathbf{(\underline{15})}$ holds by Eq-Comm and A-Alias'. Hence, by R-Trans, (13), (15), and (14), we get $\forall\,(\alpha = \sigma_0)\ C_b(\sigma_0) \mathrel{-\!\!\!\!\sqsubseteq}^{\emptyset} \forall\,(\alpha = \sigma_0)\ C_b(\alpha)$, which is the expected result (4).

The proof of the equivalence between $\sqsubseteq$ and $\sqsubseteq^{\bar{\alpha}}$ is similar. A new proof for the case I-Abstract is needed, though, since the induction hypothesis cannot be used (indeed, $\mathrel{-\!\!\!\!\sqsubseteq}$ is equal to $\mathrel{-\!\!\!\!\sqsubseteq}^{\emptyset}$, but not in general to $\mathrel{-\!\!\!\!\sqsubseteq}^{\bar{\alpha}}$). ■

Proof of Lemma 2.5.7

By induction on the derivation of $(Q)\ \sigma_1 \diamondsuit^{\bar{\alpha}} \sigma_2$.

- Case A-Equiv': by Corollary 1.5.10 and Property 1.5.11.ix (page 54).
- Case R-Trans The premises are $(Q)\ \sigma_1 \diamondsuit^{\bar{\alpha}} \sigma'_1$ $\mathbf{(\underline{1})}$ and $(Q)\ \sigma'_1 \diamondsuit^{\bar{\alpha}} \sigma_2$ $\mathbf{(\underline{2})}$. We have $\forall\,(Q)\ \sigma_1/ \leqslant_{/} \forall\,(Q)\ \sigma'_1/$ $\mathbf{(\underline{3})}$ by Property 2.1.3.ii (page 65), and $\forall\,(Q)\ \sigma'_1/ \leqslant_{/} \forall\,(Q)\ \sigma_2/$ $\mathbf{(\underline{4})}$ Since $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$ (by hypothesis), (3) become $\forall\,(Q)\ \sigma_2/ \leqslant_{/} \forall\,(Q)\ \sigma'_1/$ $\mathbf{(\underline{5})}$. By (4), (5) and antisymmetry of $\leqslant$ (Property 2.1.2.i (page 65)), we get $\forall\,(Q)\ \sigma_2/ = \forall\,(Q)\ \sigma'_1/$. Similarly, $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma'_1/$ $\mathbf{(\underline{6})}$ holds. If $\alpha \in \mathsf{ftv}(\widehat{Q}(\sigma_2))$ and $\alpha \in \bar{\alpha}$, then $\alpha \in \mathsf{ftv}(\widehat{Q}(\sigma'_1))$ by induction hypothesis on (2), thus $\alpha \in \mathsf{ftv}(\widehat{Q}(\sigma_1))$ by induction hypothesis on (1).
- Case R-Context-R': We have $\sigma_1 = \forall\,(\beta \diamond \sigma)\ \sigma'_1$ and $\sigma_2 = \forall\,(\beta \diamond \sigma)\ \sigma'_2$. We choose β fresh, that is, $\beta \notin \mathsf{dom}(\widehat{Q}) \cup \mathsf{codom}(\widehat{Q})$. The premise is $(Q, \beta \diamond \sigma)\ \sigma'_1 \diamondsuit^{\bar{\alpha} \cup \{\beta\}} \sigma'_2$ $\mathbf{(\underline{1})}$. By hypothesis, we have $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$, that is, $\forall\,(Q, \beta \diamond \sigma)\ \sigma'_1/ = \forall\,(Q, \beta \diamond \sigma)\ \sigma'_2/$. Let α be in $\bar{\alpha}$ and in $\mathsf{ftv}(\widehat{Q}(\sigma_2))$ $\mathbf{(\underline{2})}$. If α is in $\mathsf{ftv}(\widehat{Q}(\sigma'_2))$, then by induction hypothesis on (1), α is in $\mathsf{ftv}(\widehat{Q}(\sigma'_1))$, thus $\alpha \in \mathsf{ftv}(\widehat{Q}(\sigma_1))$. Otherwise, we have $\alpha \notin \mathsf{ftv}(\widehat{Q}(\sigma'_2))$ and (2), thus we necessarily have $\beta \in \mathsf{ftv}(\sigma'_2)$ $\mathbf{(\underline{3})}$. and $\alpha \in \mathsf{ftv}(\widehat{Q}(\sigma))$ $\mathbf{(\underline{4})}$. Let θ' be $\widehat{Q, \beta \diamond \sigma}$. We proceed by case analysis.

Subcase $\sigma \notin \mathcal{T}$: Then θ' is $\widehat{Q}$ $\mathbf{(\underline{5})}$, thus we have $\beta \in \mathsf{ftv}(\theta'(\sigma'_2))$ from (3), and β is obviously in $\bar{\alpha} \cup \{\beta\}$. By induction hypothesis and (1), we get $\beta \in \mathsf{ftv}(\theta'(\sigma'_1))$. Consequently, $\beta \in \mathsf{ftv}(\widehat{Q}(\sigma'_1))$ $\mathbf{(\underline{6})}$ by (5), thus $\alpha \in \mathsf{ftv}(\widehat{Q}(\sigma_1))$ by (6) and (4). This is the expected result.

Subcase $\sigma \in \mathcal{T}$: Then $\sigma \equiv \tau$ $\mathbf{(\underline{7})}$ by Property 1.5.11.ii (page 54) and θ' is $\widehat{Q} \circ [\beta = \tau]$ $\mathbf{(\underline{8})}$. We have $\alpha \in \mathsf{ftv}(\widehat{Q}(\tau))$ from (4) and (7), thus $\alpha \in \mathsf{ftv}(\theta'(\sigma'_2))$ by (8) and (3). Hence, by induction hypothesis and (1), we have $\alpha \in \mathsf{ftv}(\theta'(\sigma'_1))$. Consequently, $\alpha \in \mathsf{ftv}(\widehat{Q}(\forall\,(\beta \diamond \sigma)\ \sigma'_1))$ by (8) and (7). This is the expected result.

∘ Case A-Context-L' and I-Context-L': We have $\sigma_1 = \forall\,(\beta \diamond \sigma'_1)\ \sigma$ and $\sigma_2 = \forall\,(\beta \diamond \sigma'_2)\ \sigma$. The premise is $(Q)\ \sigma'_1 \Diamond^{\bar{\alpha}} \sigma'_2$ $\mathbf{(\underline{1})}$. We have $\alpha \in \mathsf{ftv}(\widehat{Q}(\sigma_2))$ by hypothesis. If $\alpha \in \mathsf{ftv}(\widehat{Q}(\sigma))$, then $\alpha \in \mathsf{ftv}(\widehat{Q}(\sigma_1))$, which is the expected result. Otherwise, we must have $\alpha \in \mathsf{ftv}(\widehat{Q}(\sigma'_2))$ $\mathbf{(\underline{2})}$ and $\beta \in \mathsf{ftv}(\sigma)$ $\mathbf{(\underline{3})}$. By definition, there exists u such that $\sigma/u = \beta$. By hypothesis, $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$. Hence, $\forall\,(Q)\ \sigma_1 \cdot u/ = \forall\,(Q)\ \sigma_2 \cdot u/$. By Property 1.3.3.i (page 40), this means $\Theta_Q(\sigma_1) \cdot u/ = \Theta_Q(\sigma_2) \cdot u/$, that is, $\Theta_Q(\sigma'_1) \cdot \epsilon/ = \Theta_Q(\sigma'_2) \cdot \epsilon/$. By Property 1.3.3.i (page 40), we get $\forall\,(Q)\ \sigma'_1/ = \forall\,(Q)\ \sigma'_2/$. By induction hypothesis,(1), and (2), we get $\alpha \in \mathsf{ftv}(\widehat{Q}(\sigma'_1))$, thus $\alpha \in \mathsf{ftv}(\widehat{Q}(\sigma_1))$ by (3).

∘ Case I-Rigid': Both sides have the same set of free variables.

∘ Case A-Hyp' and I-Hyp': Here σ_2 is β, and β is not in $\bar{\alpha}$, hence there is no $\alpha \in \mathsf{ftv}(\sigma_2)$ such that α is in $\bar{\alpha}$.

∘ Case A-Alias' and I-Alias': Both sides have the same set of free variables.

∘ Case A-Up' and I-Up': Both sides have the same set of free variables.

∘ Case I-Nil': We have $(Q)\ \bot \sqsubseteq \sigma_2$ and $\forall\,(Q)\ \bot/ = \forall\,(Q)\ \sigma_2/$. Hence, σ_2 is either $\bot$, or a type variable. This is a contradiction with restrictions of Lemma 2.5.6. Hence, this case does not happen. ■

Proof of Property 2.6.2

We are glad to prove each point.
Property i: it is easy to check that $\dot{\sqsubset\!\!\!-}^{\bar{\alpha}}$ is included in $\sqsubset\!\!\!-$ (indeed, rules StSh-Hyp, StSh-Up and StSh-Alias are derivable with $\sqsubset\!\!\!-$). Additionally, $\equiv$ is included in $\sqsubset\!\!\!-$, and $\sqsubset\!\!\!-$ is transitive, thus $(\equiv\dot{\sqsubset\!\!\!-}^{\emptyset})^*$ is included in $\sqsubset\!\!\!-$. Conversely, we show that $\sqsubset\!\!\!-$ is included in $(\equiv\dot{\sqsubset\!\!\!-}^{\emptyset})^*$.

As seen in Lemma 2.5.5, the relations $\sqsubset\!\!\!-$ and $\sqsubset\!\!\!-^{\bar{\alpha}}$ are equivalent. Hence, it suffices to show that $\sqsubset\!\!\!-^{\bar{\alpha}}$ is included in $(\equiv\dot{\sqsubset\!\!\!-}^{\emptyset})^*$. By Lemma 2.5.6, we can assume that the given derivation is restricted. As we did in Lemma 2.5.1 (page 78), transitivity can be lifted to top-level for $\sqsubset\!\!\!-^{\bar{\alpha}}$ too, thus it suffices to show that $\sqsubset\!\!\!-^{\bar{\alpha}}$ without transitivity is included in $(\equiv\dot{\sqsubset\!\!\!-}^{\emptyset})$. More precisely, we prove that if we have a restricted derivation of $(Q)\ \sigma_1 \sqsubset\!\!\!-^{\bar{\alpha}} \sigma_2$ without transitivity, then $(Q)\ \sigma_1\ (\equiv\dot{\sqsubset\!\!\!-}^{\emptyset})\ \sigma_2$ is derivable. As we did in the proof of Lemma 2.5.2 (page 78), we can write the derivation of $(Q)\ \sigma_1 \sqsubset\!\!\!-^{\bar{\alpha}} \sigma_2$ in the form

$$\dfrac{\dfrac{\cdots}{(QQ')\ \sigma'_1 \mathrel{\sqsubset\!\!\!\!-}^{\bar\alpha} \sigma'_2}\ \text{A-X}}{\dfrac{\vdots}{(Q)\ \sigma_1 \mathrel{\sqsubset\!\!\!\!-}^{\bar\alpha} \sigma_2}}$$

Where, A-X is either A-Equiv, A-Up', A-Hyp' or A-Alias', and the symbol $\vdots$ represents a sequence of context rules. If A-X is A-Equiv, then $(Q)\ \sigma_1 \equiv \sigma_2$ holds, thus $(Q)\ \sigma_1\ (\equiv\mathrel{\dot\sqsubset\!\!\!\!-}^{\emptyset})\ \sigma_2$ holds by definition. Otherwise, as we did in the proof of Lemma 2.5.2 (page 78), we can associate a rigid context C_r to the sequence of context rules ($\vdots$), such that $Q' = \overline{C_r}$, $\sigma_1 = C_r(\sigma'_1)$ and $\sigma_2 = C_r(\sigma'_2)$. In summary, we have $(Q)\ C_r(\sigma'_1) \mathrel{\sqsubset\!\!\!\!-}^{\bar\alpha} C_r(\sigma'_2)$, and we must show that $(Q)\ C_r(\sigma'_1)\ (\equiv\mathrel{\dot\sqsubset\!\!\!\!-}^{\emptyset})\ C_r(\sigma'_2)$ holds.

∘ Case Rule A-X is A-Hyp': Then we have $(\alpha = \sigma'_1) \in Q$ (since $\alpha \notin \bar\alpha$) and $\sigma'_2 = \alpha$. Hence, $(Q)\ C_r(\sigma'_1) \mathrel{\dot\sqsubset\!\!\!\!-}^{\bar\alpha} C_r(\sigma'_2)$ holds by Rule StSh-Hyp whenever $\sigma'_1 \notin \mathcal{T}$. If $\sigma'_1 \in \mathcal{T}$, then $(Q)\ C_r(\sigma'_1) \equiv C_r(\alpha)$ is easily derivable by Eq-Mono, thus $(Q)\ C_r(\sigma'_1)\ (\equiv\mathrel{\dot\sqsubset\!\!\!\!-}^{\emptyset})\ C_r(\sigma'_2)$ holds.

∘ Case Rule A-X is A-Up': Then we have $\sigma'_1 = \forall\,(\alpha = \forall\,(\alpha_1 = \sigma_1)\ \sigma)\ \sigma'$ and $\sigma'_2 = \forall\,(\alpha_1 = \sigma_1)\ \forall\,(\alpha = \sigma)\ \sigma'$. Hence, $(Q)\ \sigma_1 \mathrel{\dot\sqsubset\!\!\!\!-}^{\bar\alpha} \sigma_2$ holds by Rule StSh-Up (all side conditions obviously hold since Q' is $\emptyset$ and the rule is restricted).

∘ Case Rule A-X is A-Alias': Then we have $\sigma'_1 = \forall\,(\alpha_1 = \sigma_0)\ \forall\,(\alpha_2 = \sigma_0)\ \sigma$ and $\sigma'_2 = \forall\,(\alpha_1 = \sigma_0)\ \forall\,(\alpha_2 = \alpha_1)\ \sigma$. Hence, $(Q)\ C_r(\sigma'_1) \mathrel{\dot\sqsubset\!\!\!\!-}^{\bar\alpha} C_r(\sigma'_2)$ holds by StSh-Alias (because the judgment is restricted, the side conditions hold).

We have shown that $(Q)\ \sigma_1\ (\equiv\mathrel{\dot\sqsubset\!\!\!\!-}^{\emptyset})\ \sigma_2$ holds, thus $\mathrel{\sqsubset\!\!\!\!-}$ is included in $(\equiv\mathrel{\dot\sqsubset\!\!\!\!-}^{\emptyset})^*$. Consequently, $\mathrel{\sqsubset\!\!\!\!-}$ and $(\equiv\mathrel{\dot\sqsubset\!\!\!\!-}^{\emptyset})^*$ are equivalent. ☐

Property ii: We have $\mathrel{\sqsubset\!\!\!\!-}$ included in $\dot{\underline{\sqsubseteq}}$ by C-Abstract-R. Hence, $\mathrel{\sqsubset\!\!\!\!-}$ is included in $(\equiv\dot{\underline{\sqsubseteq}})^*$ (**1**).

We introduce a derivable rule:

The following rule is derivable:

$$\text{C-Abstract}\ \dfrac{(QQ')\ \sigma_1 \mathrel{\sqsubset\!\!\!\!-}^{\mathsf{dom}(Q')} \sigma_2 \qquad Q' = \overline{C_f}}{(Q)\ C_f(\sigma_1)\ (\equiv\dot{\underline{\sqsubseteq}})^*\ C_f(\sigma_2)}$$

Proof: We assume $(QQ')\ \sigma_1 \mathrel{\sqsubset\!\!\!\!-}^{\bar\alpha} \sigma_2$ (**2**) holds. If $\mathsf{level}(C_f) = 0$, then C_f is useless by Property 2.4.3.iii (page 73), thus $C_f(\sigma_1) \equiv C_f(\sigma_2)$ by definition, and $(Q)\ C_f(\sigma_1)\ (\equiv\dot{\underline{\sqsubseteq}})\ C_f(\sigma_2)$ holds. If $\mathsf{level}(C_f)$ is 1, then C_f is narrow by Property 2.4.4.i (page 74) thus C_f is also a rigid context. Hence, $(Q)\ C_f(\sigma_1) \mathrel{\sqsubset\!\!\!\!-}^{\bar\alpha} C_f(\sigma_2)$ holds by context rules. Thus $(Q)\ C_f(\sigma_1)\ (\equiv\dot{\sqsubseteq})^*\ C_f(\sigma_2)$ holds by Lemma 2.5.5 and (1). Otherwise, $\mathsf{level}(C_f) > 1$. By Lemma 2.5.5, Property i, and (2), we have $(QQ')\ \sigma_1\ (\equiv\dot{\boxminus}^{\emptyset})^*\ \sigma_2$. Since $(\equiv\dot{\sqsubseteq})^*$ is transitive, it suffices to show that if $(QQ')\ \sigma_a\ (\equiv\dot{\boxminus}^{\emptyset})\ \sigma_b$ holds, then $(Q)\ C_f(\sigma_a)\ (\equiv\dot{\sqsubseteq})^*\ C_f(\sigma_b)$ holds too. Additionally, since $\equiv$ is included in $(\equiv\dot{\sqsubseteq})$, it suffices to show that if $(QQ')\ \sigma_a\ \dot{\boxminus}^{\bar{\alpha}}\ \sigma_b$ holds, then $(Q)\ C_f(\sigma_a)\ (\equiv\dot{\sqsubseteq})^*\ C_f(\sigma_b)$ holds too. We get such a result by C-Abstract-F. ☐

The relation $\dot{\sqsubset}$ is included in $\sqsubseteq$, as well as $\equiv$ and $\dot{\boxminus}^{\bar{\alpha}}$. Besides, $\sqsubseteq$ is a congruence under flexible prefixes C_f. Hence, $\dot{\sqsubseteq}$ is included in $\sqsubseteq$. Since $\sqsubseteq$ is transitive, $\dot{\sqsubseteq}^*$ is also included in $\sqsubseteq$. Additionally, $\equiv$ is included in $\sqsubseteq$ by I-Equiv*, thus $(\equiv\dot{\sqsubseteq})^*$ is included in $\sqsubseteq$.

Conversely, we show that $\sqsubseteq$ is included in $(\equiv\dot{\sqsubseteq})^*$. By Lemma 2.5.5, it suffices to show that $\sqsubseteq^{\bar{\alpha}}$ is included in $(\equiv\dot{\sqsubseteq})^*$. As shown in Lemma 2.5.1 (page 78), transitivity can be lifted at top-level, and the derivation is made restricted by Lemma 2.5.6; thus, we only have to show that if we have a restricted derivation of $(Q)\ \sigma_1 \sqsubseteq^{\bar{\alpha}} \sigma_2$, without using transitivity, then $(Q)\ \sigma_1\ (\equiv\dot{\sqsubseteq})^*\ \sigma_2$ holds too. The derivation of $(Q)\ \sigma_1 \sqsubseteq^{\bar{\alpha}} \sigma_2$ is of the form

$$\begin{array}{rc} & \overline{\phantom{(QQ')\ \sigma_1' \sqsubseteq^{\bar{\alpha}} \sigma_2'}} \\[-6pt] \text{I-X} & \dfrac{\quad}{(QQ')\ \sigma_1' \sqsubseteq^{\bar{\alpha}} \sigma_2'} \\ & \vdots \\ \text{I-Context-X'} & \dfrac{\quad}{(Q)\ \sigma_1 \sqsubseteq^{\bar{\alpha}} \sigma_2} \end{array}$$

The context rules (represented by $\vdots$) are associated to a flexible context C_f such that $\overline{C_f} = Q'$, $\sigma_1 = C_f(\sigma_1')$ and $\sigma_2 = C_f(\sigma_2')$. The top Rule I-X is I-Abstract', I-Bot', I-Hyp', I-Rigid', I-Up', or I-Alias'.

- Case I-Abstract': By hypothesis $(QQ')\ \sigma_1'\ \boxminus^{\mathsf{dom}(Q')}\ \sigma_2'$ holds. We have $(Q)\ \sigma_1\ (\equiv\dot{\sqsubseteq})^*\ \sigma_2$ by C-Abstract. This is the expected result.
- Case I-Hyp': By rules S-Hyp and C-Strict.
- Case I-Rigid': By rules S-Rigid and C-Strict.
- Case I-Bot': We have $\sigma_1' = \bot$. If σ_2' is closed, we conclude directly by S-Nil and C-Strict. Otherwise, we consider $\forall\,(QQ')\ \sigma_2'$, which is necessarily closed. We can derive $(Q)\ \sigma_1\ \dot{\sqsubseteq}\ C_f(\forall\,(QQ')\ \sigma_2')$ by S-Nil and C-Strict. By iteration of S-Hyp and C-Strict, or A-Hyp and C-Abstract, we derive $(Q)\ C_f(\forall\,(QQ')\ \sigma_2')\ \dot{\sqsubseteq}^*\ C_f(\sigma_2')$. Hence, $(Q)\ \sigma_1\ \dot{\sqsubseteq}^*\ C_f(\sigma_2')$ holds by transitivity, that is $(Q)\ \sigma_1\ \dot{\sqsubseteq}^*\ \sigma_2$.
- Case I-Up': By S-Up and C-Strict.
- Case I-Alias': By S-Alias and C-Strict.

■

Proof of Property 2.6.3

Each point deserves its own proof.

Property i for $\mathcal{R}$ being $\dot{\sqsubseteq}^{\bar\alpha}$: By case on the rule used to derive $(Q)\ \sigma_1 \dot{\sqsubseteq}^{\bar\alpha} \sigma_2$.

∘ Case StSh-Hyp: By hypothesis, we have $\sigma_1 = C_r(\sigma'_1)$, $(\alpha = \sigma''_1) \in Q$ **(1)**, $\sigma_2 = C_r(\alpha)$ and $(Q)\ \widehat{C_r}(\sigma'_1) \equiv \sigma''_1$ **(2)**. Besides, $\sigma'_1 \notin \mathcal{T}$ and C_r is a useful well-formed context by hypothesis. By Lemma 2.4.5 (page 75), there exists a context C'_r, which is $\mathsf{ftv}(\widehat{C_r}(\sigma'_1))$-equivalent to C_r, such that $\mathsf{nf}(\sigma_1) = C'_r(\sigma_f)$ **(3)**, where σ_f is $\mathsf{nf}(\widehat{C_r}(\sigma'_1))$. Hence, $\sigma_f \equiv \widehat{C_r}(\sigma'_1)$ **(4)** by Property 1.5.6.i (page 51). Consequently, (2) and (4) lead to $(Q)\ \sigma_f \equiv \sigma''_1$ **(5)**. We have $(Q)\ \mathsf{nf}(\sigma_1) \dot{\sqsubseteq}^{\bar\alpha} C'_r(\alpha)$ **(6)** by StSh-Hyp, (3), (1), and (5). Besides, $(Q)\ C'_r(\alpha) \equiv C_r(\alpha)$ **(7)** holds since C'_r and C_r are $\mathsf{ftv}(\widehat{C_r}(\sigma'_1))$-equivalent and $\alpha \notin \mathsf{dom}(C_r) \cup \mathsf{dom}(C'_r)$. We have the expected result, namely (6) and (7), taking $\sigma'_2 = C'_r(\alpha)$.

∘ Case StSh-Up: By hypothesis, we have $\sigma_1 = C_r(\forall\,(\beta = \forall\,(Q', \alpha_0 = \sigma_0)\ \sigma')\ \sigma'')$, and $\sigma_2 = C_r(\forall\,(\alpha_0 = \sigma_0)\ \forall\,(\beta = \forall\,(Q')\ \sigma')\ \sigma'')$ **(8)**. along with a bunch of side-conditions. Let σ_a be $\forall\,(\beta = \forall\,(Q', \alpha_0 = \sigma_0)\ \sigma')\ \sigma''$. By hypothesis we have $\sigma_0 \notin \mathcal{T}$, $\alpha_0 \in \mathsf{ftv}(\sigma')$ and $\beta \in \mathsf{ftv}(\sigma'')$, thus $\sigma_a \notin \mathcal{T}$ by Property 2.1.5.i (page 67). Let θ be $\widehat{C_r}$ and $\bar\alpha$ be $\mathsf{ftv}(\theta(\sigma_a))$. By hypothesis, C_r is a useful well-formed context, thus, by Lemma 2.4.5 (page 75), there exists a context C'_r, that is $\bar\alpha$-equivalent to C_r, such that $\mathsf{nf}(\sigma_1) = C'_r(\sigma'_1)$ **(9)**, with $\sigma'_1 = \mathsf{nf}(\theta(\sigma_a)))$. Hence, by Property 1.5.6.iii (page 51), we have $\sigma'_1 = \theta(\forall\,(\beta = \sigma''_1)\ \mathsf{nf}(\sigma''))$ **(10)**, where σ''_1 is $\mathsf{nf}(\forall\,(Q')\ \forall\,(\alpha_0 = \sigma_0)\ \sigma')$. We write θ' for $\widehat{Q'}$. By definition of normal form σ''_1 is $\forall\,(Q'')\ \forall\,(\alpha_0 = \mathsf{nf}(\sigma_0))\ \theta'(\mathsf{nf}(\sigma'))$ **(11)**, with $\forall\,(Q'')\ \theta'(\mathsf{nf}(\sigma')) = \mathsf{nf}(\forall\,(Q')\ \sigma')$ **(12)**. Combining (9), (10), and (11), $\mathsf{nf}(\sigma_1)$ is (you may take a deep breath) $C'_r(\forall\,(\beta = \forall\,(\theta(Q''))\ \forall\,(\alpha_0 = \theta(\mathsf{nf}(\sigma_0)))\ \theta\theta'(\mathsf{nf}(\sigma')))\ \theta(\mathsf{nf}(\sigma'')))$. In addition, all side conditions of StSh-Up are satisfied. Hence, StSh-Up can be applied: $(Q)\ \mathsf{nf}(\sigma_1) \dot{\sqsubseteq}^{\bar\alpha} C'_r(\forall\,(\alpha_0 = \theta(\mathsf{nf}(\sigma_0)))\ \forall\,(\beta = \forall\,(\theta(Q''))\ \theta\theta'(\mathsf{nf}(\sigma')))\ \theta(\mathsf{nf}(\sigma'')))$.

Now, watchful readers expect a proof that $C'_r(\forall\,(\alpha_0 = \theta(\mathsf{nf}(\sigma_0)))\ \forall\,(\beta = \forall\,(\theta(Q''))\ \theta\theta'(\mathsf{nf}(\sigma')))\ \theta(\mathsf{nf}(\sigma'')))$ is equivalent to σ_2. Let σ'_2 be the former. We get $\sigma'_2 \equiv C'_r(\forall\,(\alpha_0 = \theta(\sigma_0))\ \forall\,(\beta = \theta(\forall\,(Q')\ \sigma'))\ \theta(\sigma''))$ from (12) and Property 1.5.6.i (page 51). As for σ_2, we use Eq-Mono and derive $\sigma_2 \equiv C_r(\forall\,(\alpha_0 = \theta(\sigma_0))\ \forall\,(\beta = \theta(\forall\,(Q')\ \sigma'))\ \theta(\sigma''))$ from (8). Hence, $\sigma_2 \equiv C_r(\sigma_3)$ and $\sigma'_2 \equiv C'_r(\sigma_3)$, where σ_3 is $\forall\,(\alpha_0 = \theta(\sigma_0))\ \forall\,(\beta = \theta(\forall\,(Q')\ \sigma'))\ \theta(\sigma'')$. We know that C_r and C'_r are $\bar\alpha$-equivalent. Moreover, $\mathsf{ftv}(\sigma_3) = \mathsf{ftv}(\theta(\mathsf{ftv}(\sigma_a))) = \bar\alpha$, thus by definition of $\bar\alpha$-equivalence, we have $C_r(\sigma_3) \equiv C'_r(\sigma_3)$, which gives $\sigma_2 \equiv \sigma'_2$. This is the expected result.

∘ Case StSh-Alias: We have $\sigma_1 = C_r(\forall\,(\alpha_a = \sigma_a)\ \forall\,(Q')\ \forall\,(\alpha_b = \sigma_b)\ \sigma')$ with $\sigma_a \equiv \sigma_b$, and σ_2 is $C_r(\forall\,(\alpha_a = \sigma_a)\ \forall\,(Q')\ \forall\,(\alpha_b = \alpha_a)\ \sigma')$. Let σ_0 be $\forall\,(\alpha_a = \sigma_a)\ \forall\,(Q')\ \forall\,(\alpha_b = \sigma_b)\ \sigma'$ **(13)**. Let θ be $\widehat{C_r}$ and $\bar\alpha$ be $\mathsf{ftv}(\theta(\sigma_0))$. By hypothesis, $\sigma_a \notin \mathcal{T}$ and

$\alpha_a \in \mathsf{ftv}(\sigma')$, thus $\sigma_0 \notin \mathcal{T}$ by Property 2.1.5.i (page 67). Besides, C_r is a useful well-formed context, thus by Lemma 2.4.5 (page 75), there exists a context C'_r which is $\bar{\alpha}$-equivalent to C_r such that $\mathsf{nf}(\sigma_1) = C'_r(\mathsf{nf}(\theta(\sigma_0)))$. By Property 1.5.6.iii (page 51), we have $\mathsf{nf}(\sigma_1) = C'_r(\theta(\mathsf{nf}(\sigma_0)))$ $(\underline{\mathbf{14}})$.

Let θ' be $\widehat{Q'}$. From (13), $\mathsf{nf}(\sigma_0)$ is of the form $\forall\,(\alpha_a = \mathsf{nf}(\sigma_a))\;\forall\,(Q'')\;\forall\,(\alpha_b = \mathsf{nf}(\sigma_b))\;\theta'(\mathsf{nf}(\sigma'))$ $(\underline{\mathbf{15}})$. Consequently, from (14) and (15), $\mathsf{nf}(\sigma_1)$ is $C'_r(\forall\,(\alpha_a = \theta(\mathsf{nf}(\sigma_a)))\;\forall\,(\theta(Q''))\;\forall\,(\alpha_b = \theta(\mathsf{nf}(\sigma_b)))\;\theta\theta'(\mathsf{nf}(\sigma')))$ $(\underline{\mathbf{16}})$, besides, $\alpha_a, \alpha_b \in \mathsf{ftv}(\theta\theta'(\mathsf{nf}(\sigma')))$ $(\underline{\mathbf{17}})$ by Property 1.5.6.ii (page 51). By hypothesis $\sigma_a \notin \mathcal{T}$, hence $\theta(\mathsf{nf}(\sigma_a)) \notin \mathcal{T}$ $(\underline{\mathbf{18}})$ by property 1.5.11.iv (page 54). Moreover $\sigma_a \equiv \sigma_b$, thus $\mathsf{nf}(\sigma_a) \equiv \mathsf{nf}(\sigma_b)$ by Property 1.5.6.i (page 51) and R-Trans. This gives $\theta(\mathsf{nf}(\sigma_a)) \equiv \theta(\mathsf{nf}(\sigma_b))$ $(\underline{\mathbf{19}})$ by Property 1.5.11.v (page 54). Consequently, from (16), (17), (18), (19), and StSh-Alias, we can derive $(Q)\;\mathsf{nf}(\sigma_1) \mathrel{\dot{\sqsubseteq}^{\bar{\alpha}}} \sigma'_2$, where σ'_2 is $C'_r(\forall\,(\alpha_a = \theta(\mathsf{nf}(\sigma_a)))\;\forall\,(\theta(Q''))\;\forall\,(\alpha_b = \alpha_a)\;\theta\theta'(\mathsf{nf}(\sigma')))$.

It remains to be shown that $\sigma_2 \equiv \sigma'_2$ holds. We have $\sigma_2 \equiv C_r(\forall\,(\alpha_a = \theta(\mathsf{nf}(\sigma_a)))\;\forall\,(\theta(Q''))\;\forall\,(\alpha_b = \alpha_a)\;\theta\theta'(\mathsf{nf}(\sigma')))$ using Eq-Mono* on $\widehat{C_r}$ and $\widehat{Q'}$. Hence, $\sigma_2 \equiv C_r(\sigma_z)$ and σ'_2 is $C'_r(\sigma_z)$, where σ_z is $\forall\,(\alpha_a = \theta(\mathsf{nf}(\sigma_a)))\;\forall\,(\theta(Q''))\;\forall\,(\alpha_b = \alpha_a)\;\theta\theta'(\mathsf{nf}(\sigma'))$. By Property 1.5.4.i (page 50), we have $\mathsf{ftv}(\sigma_z) = \theta(\mathsf{ftv}(\sigma_0))$, thus we get $\mathsf{ftv}(\sigma_z) = \bar{\alpha}$. Hence, by definition of $\bar{\alpha}$-equivalence, we get $C_r(\sigma_z) \equiv C'_r(\sigma_z)$, which leads to $\sigma_2 \equiv \sigma'_2$. This is the expected result.

Property ii for $\mathcal{R}$ being $\dot{\sqsubseteq}^{\bar{\alpha}}$: By hypothesis, $\sigma_1 \approx \sigma'_1$ $(\underline{\mathbf{1}})$ and $(Q)\;\sigma_1 \mathrel{\dot{\sqsubseteq}^{\bar{\alpha}}} \sigma_2$ $(\underline{\mathbf{2}})$ hold. We prove the result for a single commutation only, then we get the expected result by immediate induction. We assume that (1) holds by a single commutation. More precisely, let σ_r, σ'_r and C be such that

$$\sigma_r = \forall\,(\gamma = \sigma_a, \beta = \sigma_b)\;\sigma_c \qquad \sigma'_r = \forall\,(\beta = \sigma_b, \gamma = \sigma_a)\;\sigma_c \qquad \sigma_1 = C(\sigma_r)\;(\underline{\mathbf{3}})$$

$$\sigma'_1 = C(\sigma'_r) \qquad \beta \notin \mathsf{ftv}(\sigma_a) \qquad \gamma \notin \mathsf{ftv}(\sigma_b)\;(\underline{\mathbf{4}}).$$

Since σ_1 is in normal form, we also know that $\gamma, \beta \in \mathsf{ftv}(\sigma_c)$. We prove the result by case on the rule used to derive (2).

∘ Case StSh-Hyp: σ_1 is of the form $C_r(\sigma')$ and $(\alpha = \sigma_0) \in Q$ with $(Q)\;\sigma_0 \equiv \widehat{C_r}(\sigma')$. Since σ_1 is in normal form, we have $\widehat{C_r} = \mathtt{id}$. By (3), we have $C(\forall\,(\gamma = \sigma_a, \beta = \sigma_b)\;\sigma_c) = C_r(\sigma')$. By alpha-conversion, we can freely assume that $\gamma \notin \mathsf{ftv}(\sigma_0)$. We proceed by case analysis on C and C_r.

Subcase C and C_r are disjoint: By Lemma 2.4.6 (page 76), there exists a two-hole context C^2 such that $C_r = C^2([\,], \forall\,(\gamma = \sigma_a, \beta = \sigma_b)\;\sigma_c)$ and $C = C^2(\sigma', [\,])$. Besides, $C'_r = C^2([\,], \forall\,(\beta = \sigma_b, \gamma = \sigma_a)\;\sigma_c)$ $(\underline{\mathbf{5}})$ is rigid. We have $\sigma'_1 = C'_r(\sigma')$. Let σ'_2 be $C'_r(\alpha)$ $(\underline{\mathbf{6}})$. Note also that σ_2 is $C^2(\alpha, \forall\,(\gamma = \sigma_a, \beta = \sigma_b)\;\sigma_c)$ $(\underline{\mathbf{7}})$. By StSh-Hyp, we get $(Q)\;\sigma'_1 \mathrel{\dot{\sqsubseteq}^{\bar{\alpha}}} \sigma'_2$, and $\sigma'_2 \approx \sigma_2$ by (6), (5), and (7). This is the expected result.

Subcase C is of the form $C_r(C')$: We have $\sigma' = C'(\sigma_r)$, $(Q)\;\sigma_0 \equiv C'(\sigma'_r)$, and $(Q)\;\alpha \equiv \alpha$. We get the expected result by StSh-Hyp.

Subcase C_r is of the form $C(\forall\,(\gamma = C', \beta = \sigma_b)\ \sigma_c)$: immediate

Subcase C_r is of the form $C(\forall\,(\gamma = \sigma_a, \beta = C')\ \sigma_c)$: immediate

Subcase C_r is of the form $C(\forall\,(\gamma = \sigma_a, \beta = \sigma_b)\ C')$: immediate

Subcase C_r is of the form $C(\forall\,(\gamma{=}\sigma_a)\ [\,])$: We must have $\sigma_1 = C(\forall\,(\gamma{=}\sigma_a, \beta{=}\sigma_b)\ \sigma_c)$ and $(Q)\ \sigma_0 \equiv \forall\,(\beta = \sigma_b)\ \sigma_c$. However, $\gamma \in \mathsf{ftv}(\sigma_c)$, thus $\gamma \in \mathsf{ftv}(\forall\,(\beta = \sigma_b)\ \sigma_c)$. Hence, by Property 1.5.4.i (page 50), we must have $\gamma \in \mathsf{ftv}(\sigma_0)$, which is a contradiction. Hence this case cannot occur.

○ Case StSh-Up: σ_1 is of the form $C_r(\forall\,(\alpha_b{=}\forall\,(Q', \alpha_a{=}\sigma)\ \sigma')\ \sigma'')$, with $\sigma \notin \mathcal{T}$, $\alpha_a \in \mathsf{ftv}(\sigma')$, $\sigma', \sigma'' \notin \mathcal{V}$ $\textbf{(\underline{8})}$, $\alpha_a \notin \mathsf{dom}(Q\overline{C_r}Q')$ $\textbf{(\underline{9})}$, $\mathsf{ftv}(\sigma)\ \#\ \mathsf{dom}(Q')$, $\alpha_b \in \mathsf{ftv}(\sigma'')$ $\textbf{(\underline{10})}$, and σ_2 is $C_r(\forall\,(\alpha_a = \sigma)\ \forall\,(\alpha_b = \forall\,(Q')\ \sigma')\ \sigma'')$. We have $C(\forall\,(\gamma = \sigma_a, \beta = \sigma_b)\ \sigma_c) = C_r(\forall\,(\alpha_b = \forall\,(Q', \alpha_a = \sigma)\ \sigma')\ \sigma'')$. We proceed by case on the form of C and C_r:

Subcase C and C_r are disjoint: immediate

Subcase C is C_r: We have $\gamma = \alpha_b$ $\textbf{(\underline{11})}$, $\sigma_a = \forall\,(Q', \alpha_a = \sigma)\ \sigma'$ and $\sigma'' = \forall\,(\beta{=}\sigma_b)\ \sigma_c$ $\textbf{(\underline{12})}$ Then σ_1' is $C_r(\forall\,(\beta{=}\sigma_b, \gamma{=}\forall\,(Q', \alpha_a{=}\sigma)\ \sigma')\ \sigma_c)$. We have $\gamma \in \mathsf{ftv}(\sigma_c)$ by (10), (11), (12) and (4). Besides, $\sigma_c \notin \mathcal{V}$ by (12) and (8) and $\alpha_a \notin \mathsf{ftv}(\sigma_c)$ by (12) and (9). Hence, StSh-Up applies and gives $(Q)\ \sigma_1' \mathrel{\dot{\sqsubseteq}^{\bar{\alpha}}} C_r(\forall\,(\beta{=}\sigma_b, \alpha_a{=}\sigma, \gamma{=}\forall\,(Q')\ \sigma')\ \sigma_c)$. We get the expected result by observing that $C_r(\forall\,(\beta{=}\sigma_b, \alpha_a{=}\sigma, \gamma{=}\forall\,(Q')\ \sigma')\ \sigma_c)$ is equivalent to σ_2.

Subcase C is of the form $C_r(C')$: immediate.

Subcase C_r is of the form $C(C')$: immediate.

○ Case StSh-Alias: all subcases are easy.

Property i for $\mathcal{R}$ being $\dot{\sqsubset}$: By case on the rule used to derive $(Q)\ \sigma_1 \mathrel{\dot{\sqsubset}} \sigma_2$. Cases S-Hyp, S-Up and S-Alias are similar to cases StSh-Hyp, StSh-Up, and StSh-Alias of Property i.

○ Case S-Nil: We have $\sigma_1 = C_f(\bot)$, $\sigma_2 = C_f(\sigma)$ and σ is closed by hypothesis. By Lemma 2.4.5 (page 75), there exists C_r', $\emptyset$-equivalent to C_r, such that $\mathsf{nf}(\sigma_1) = C_r'(\bot)$. Consequently, $\mathsf{nf}(\sigma_1) \mathrel{\dot{\sqsubset}} C_r'(\sigma)$ holds by S-Nil. Additionally, we have $C_r(\sigma) \equiv C_r'(\sigma)$ by definition of $\emptyset$-equivalence.

○ Case S-Rigid: We have $\sigma_1 = C_f(\forall\,(\alpha \geq \sigma')\ \sigma)$ and $\sigma_2 = C_f(\forall\,(\alpha = \sigma')\ \sigma)$. By hypothesis, $\sigma \notin \mathcal{V}$, $\sigma' \notin \mathcal{T}$ $\textbf{(\underline{13})}$ and $\alpha \in \mathsf{ftv}(\sigma)$ $\textbf{(\underline{14})}$. Let σ_0 be $\forall\,(\alpha \geq \sigma')\ \sigma$, $\bar{\alpha}$ be $\mathsf{ftv}(\theta(\sigma_0))$ and θ be $\widehat{C_r}$. By Property 2.1.5.i (page 67) applied to (13) and (14), we have $\sigma_0 \notin \mathcal{T}$. By Lemma 2.4.5 (page 75), there exists C_r', $\bar{\alpha}$-equivalent to C_r, such that $\mathsf{nf}(\sigma_1) = C_r'(\mathsf{nf}(\theta(\sigma_0)))$. Hence, by Property 1.5.6.iii (page 51), $\mathsf{nf}(\sigma_1)$ is $C_r'(\theta(\forall\,(\alpha \geq \mathsf{nf}(\sigma'))\ \mathsf{nf}(\sigma)))$, that is, $C_r'(\forall\,(\alpha \geq \theta(\mathsf{nf}(\sigma')))\ \theta(\mathsf{nf}(\sigma)))$. By Rule S-Rigid, we can derive $(Q)\ \mathsf{nf}(\sigma_1) \mathrel{\dot{\sqsubset}} C_r'(\sigma_0')$, where σ_0' is $\forall\,(\alpha = \theta(\mathsf{nf}(\sigma')))\ \theta(\mathsf{nf}(\sigma))$. We note that σ_0' is $\theta(\mathsf{nf}(\forall\,(\alpha = \sigma')\ \sigma))$. Consequently, $\sigma_2 \equiv C_r(\sigma_0')$ holds by by Property 1.5.6.i (page 51) and by Rule Eq-Mono. We have $\mathsf{ftv}(\sigma_0') = \theta(\mathsf{ftv}(\sigma_0))$ by Property 1.5.6.ii (page 51), thus $\mathsf{ftv}(\sigma_0') = \bar{\alpha}$. Hence, by definition of $\bar{\alpha}$-equivalence, $C_r(\sigma_0') \equiv C_r'(\sigma_0')$.

Property ii for $\mathcal{R}$ being $\dot\sqsubset$: By case on the rule used to derive $(Q)\ \sigma_1 \mathrel{\dot\sqsubset} \sigma_2$. Cases S-Hyp, S-Up and S-Alias are similar to cases StSh-Hyp, StSh-Up, and StSh-Alias of Property ii.

- Case S-Nil: immediate.
- Case S-Rigid: immediate. ■

Proof of Lemma 2.7.8

Directly, we assume that we have

$$\widehat{Q}(\mathsf{nf}(\sigma_2)) \notin \vartheta\ (\underline{\mathbf{1}}) \qquad (Q)\ \sigma_1 \sqsubseteq \sigma_2\ (\underline{\mathbf{2}}) \qquad \forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/\ (\underline{\mathbf{3}})$$

$$X \notin w(\sigma_1) - w(\sigma_2)\ (\underline{\mathbf{4}})$$

By Lemma 2.5.5 and (2), we have a derivation of $(Q)\ \sigma_1 \sqsubseteq^{\bar\alpha} \sigma_2$ $\mathbf{(\underline{5})}$. By Lemma 2.5.6, this derivation is restricted. The proof is by induction on the derivation of (5).

- Case I-Abstract': The premise is $(Q)\ \sigma_1 \mathrel{-\!\!\!\!\!\sqsubseteq}^{\bar\alpha} \sigma_2$ and we conclude by Lemma 2.5.5.
- Case R-Trans: The premises are $(Q)\ \sigma_1 \sqsubseteq^{\bar\alpha} \sigma'_1$ $\mathbf{(\underline{6})}$ and $(Q)\ \sigma'_1 \sqsubseteq^{\bar\alpha} \sigma_2$ $\mathbf{(\underline{7})}$. By hypothesis (3), $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$ holds. Hence, $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma'_1/ = \forall\,(Q)\ \sigma_2/$ $\mathbf{(\underline{8})}$ holds (see case R-Trans in the proof of Property 2.7.6.i (page 94) for details). Besides, by Property 2.7.6.i (page 94), (6), and (7), we have $w(\sigma_1) \geqslant w(\sigma'_1) \geqslant w(\sigma_2)$ $\mathbf{(\underline{9})}$. By Property 2.7.2.iv (page 87), (4), and (9), we get $X \notin w(\sigma_1) - w(\sigma'_1)$ $\mathbf{(\underline{10})}$ and $X \notin w(\sigma'_1) - w(\sigma_2)$ $\mathbf{(\underline{11})}$. Now, by hypothesis (1), we have $\widehat{Q}(\mathsf{nf}(\sigma_2)) \notin \vartheta$. Assume $\widehat{Q}(\mathsf{nf}(\sigma'_1)) \in \vartheta$ holds. By definition, this means that we have $\widehat{Q}(\mathsf{nf}(\sigma'_1)) = \alpha$ $\mathbf{(\underline{12})}$ for some type variable α. If, by a way of contradiction, $\alpha \in \mathsf{dom}(\widehat{Q})$ holds, then, by well-formedness of Q, we must have $\alpha \notin \mathsf{codom}(\widehat{Q})$, which implies $\alpha \notin \widehat{Q}(\mathsf{nf}(\sigma'_1))$, and this is a contradiction with (12). Consequently, we have $\alpha \notin \mathsf{dom}(\widehat{Q})$ $\mathbf{(\underline{13})}$, that is, $\widehat{Q}(\alpha) = \alpha$. Then (12) gives $\widehat{Q}(\mathsf{nf}(\sigma'_1)) \equiv \widehat{Q}(\alpha)$ $\mathbf{(\underline{14})}$ by Eq-Refl. By Corollary 1.5.10 and (14), we get $(Q)\ \mathsf{nf}(\sigma'_1) \equiv \alpha$. By Property 1.5.6.i and R-Trans, this gives $(Q)\ \sigma'_1 \equiv \alpha$ $\mathbf{(\underline{15})}$. Hence, by Lemma 2.1.6, (7), and (15), we have $(Q)\ \sigma'_1 \equiv \sigma_2$ $\mathbf{(\underline{16})}$. By R-Trans, (16), and (15), we get $(Q)\ \sigma_2 \equiv \alpha$. Consequently, $\widehat{Q}(\mathsf{nf}(\sigma_2)) \in \vartheta$ holds by Corollary 1.5.10 and (13), which is a contradiction with (1). Hence, we have $\widehat{Q}(\mathsf{nf}(\sigma'_1)) \notin \mathcal{V}$ $\mathbf{(\underline{17})}$. By induction hypothesis, (6), (8), (10), and (17), we have $(Q)\ \sigma_1 \mathrel{-\!\!\!\!\!\sqsubseteq} \sigma'_1$ $\mathbf{(\underline{18})}$. By induction hypothesis, (7), (8), (11), and (1), we have $(Q)\ \sigma'_1 \mathrel{-\!\!\!\!\!\sqsubseteq} \sigma_2$ $\mathbf{(\underline{19})}$. By R-Trans, (18), and (19), we get $(Q)\ \sigma_1 \mathrel{-\!\!\!\!\!\sqsubseteq} \sigma_2$. This is the expected result.
- Case R-Context-R: We have $\sigma_1 = \forall\,(\alpha \diamond \sigma)\ \sigma'_1$ and $\sigma_2 = \forall\,(\alpha \diamond \sigma)\ \sigma'_2$. The premise is $(Q, \alpha \diamond \sigma)\ \sigma'_1 \sqsubseteq^{\bar\alpha \cup \{\alpha\}} \sigma'_2$ $\mathbf{(\underline{20})}$. By hypothesis (3), we have $\forall\,(Q, \alpha \diamond \sigma)\ \sigma'_1/ = \forall\,(Q, \alpha \diamond \sigma)\ \sigma'_2/$ $\mathbf{(\underline{21})}$. Let θ' be $\widehat{Q, \alpha \diamond \sigma}$ and θ be $\widehat{Q}$. We proceed by case analysis.

Subcase $\sigma \in \mathcal{T}$: We have $\sigma \equiv \tau$ by Property 1.5.11.ii (page 54). By definition, we have $w(\sigma_1) = w(\sigma'_1)$ and $w(\sigma_2) = w(\sigma'_2)$. Hence, we get $X \notin w(\sigma'_1) - w(\sigma'_2)$ $\mathbf{(\underline{22})}$

from (4). From (1), we have $\theta(\mathsf{nf}(\sigma'_2)[\alpha = \tau]) \notin \mathcal{V}$, which can also be written $\theta'(\mathsf{nf}(\sigma'_2)) \notin \mathcal{V}$ $\mathbf{(\underline{23})}$. By induction hypothesis, (20), (21), (22), and (23), we get $(Q, \alpha \diamond \sigma)\ \sigma'_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma'_2$. We conclude by R-CONTEXT-R.

In the following, we assume $\sigma \notin \mathcal{T}$, which implies $\theta' = \theta$. Besides, if $\alpha \in \mathsf{ftv}(\sigma'_2)$, then $\alpha \in \mathsf{ftv}(\sigma'_1)$ by Lemma 2.5.7 (page 83), (20), and (21), and $\alpha \notin \mathsf{codom}(\widehat{Q})$). Conversely, if $\alpha \in \mathsf{ftv}(\sigma'_1)$, then $\alpha \in \mathsf{ftv}(\sigma'_2)$ by Lemma 2.1.4 (page 67). Hence, we know that $\alpha \in \mathsf{ftv}(\sigma'_2)$ if and only if $\alpha \in \mathsf{ftv}(\sigma'_1)$ $\mathbf{(\underline{24})}$.

SUBCASE $\mathsf{nf}(\sigma'_1) = \alpha$: Then $\sigma'_2 \equiv \alpha$ by Property 2.1.7.ii (page 68), (24), and (20). Hence, $\sigma_1 \equiv \sigma \equiv \sigma_2$ holds by EQ-VAR, and we have $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma_2$ by A-EQUIV.

SUBCASE $\mathsf{nf}(\sigma'_2) = \alpha$: Then $\sigma'_1 \equiv \alpha$ by Property 2.1.7.ii (page 68), and we also have $(Q)\ \sigma_1 \equiv \sigma_2$. In the following, we assume that $\mathsf{nf}(\sigma'_1) \neq \alpha$ and $\mathsf{nf}(\sigma'_2) \neq \alpha$ $\mathbf{(\underline{25})}$ hold.

OTHERWISE, let x be 1 if $\alpha \in \mathsf{ftv}(\sigma'_1)$ and 0 otherwise. Let A be $X \star \diamond$. We have $w(\sigma_1) = x \times A \times w_A(\sigma) + w(\sigma'_1)$ and $w(\sigma_2) = x \times A \times w_A(\sigma) + w(\sigma'_2)$. Hence, $w(\sigma_1) - w(\sigma_2) = w(\sigma'_1) - w(\sigma'_2)$, thus $X \notin w(\sigma'_1) - w(\sigma'_2)$ $\mathbf{(\underline{26})}$ by (4). We have $\theta(\mathsf{nf}(\sigma_2)) \notin \mathcal{V}$ from (1), and $\mathsf{nf}(\sigma'_2)$ is not α from (25). Thus we have $\theta(\mathsf{nf}(\sigma'_2)) \notin \mathcal{V}$ $\mathbf{(\underline{27})}$ (otherwise, σ'_2 would be equivalent to a type variable β, and $\forall\,(\alpha \diamond \sigma)\ \sigma'_2 \equiv \beta$ would also hold, which is a contradiction with (1)). By induction hypothesis, (20), (21), (26), and (27), we get $(Q, \alpha \diamond \sigma)\ \sigma'_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma'_2$. Then $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma_2$ holds by R-CONTEXT-R. This is the expected result.

- CASE I-CONTEXT-L': We have $\sigma_1 = \forall\,(\alpha \geq \sigma'_1)\ \sigma$ and $\sigma_2 = \forall\,(\alpha \geq \sigma'_2)\ \sigma$. The premise is $(Q)\ \sigma'_1 \sqsubseteq^{\bar{\alpha}} \sigma'_2$. We have $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$ $\mathbf{(\underline{1})}$ by hypothesis, and $\mathsf{nf}(\sigma) \neq \alpha$, and $\alpha \in \mathsf{ftv}(\sigma)$ $\mathbf{(\underline{2})}$ by Lemma 2.5.6. By Property 1.3.3.i (page 40), (1), and (2), we get $\forall\,(Q)\ \sigma'_1/ = \forall\,(Q)\ \sigma'_2/$ $\mathbf{(\underline{3})}$. We have $w(\sigma_1) = X \times w(\sigma'_1) + w(\sigma)$ and $w(\sigma_2) = X \times w(\sigma'_2) + w(\sigma)$. Hence, $w(\sigma_1) - w(\sigma_2) = X \times (w(\sigma'_1) - w(\sigma'_2))$. By hypothesis, we have $X \notin w(\sigma_1) - w(\sigma_2)$. Consequently, we must have $w(\sigma'_1) - w(\sigma'_2) = 0$, that is, $w(\sigma'_1) = w(\sigma'_2)$ $\mathbf{(\underline{4})}$. By Property 2.7.6.ii (page 94), (3), and (4), we get $(Q)\ \sigma'_1 \equiv \sigma'_2$. Hence, $(Q)\ \sigma_1 \equiv \sigma_2$ holds by R-CONTEXT-L, and $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma_2$ holds by A-EQUIV.

- CASE I-BOT': We have $\sigma_1 = \bot$. Hence, $(\forall\,(Q)\ \sigma_1)/\epsilon = \bot$. By Lemma 2.5.6, we have $\sigma_2 \notin \mathcal{V}$, and σ_2 is not $\bot$. Hence, σ_2/ϵ is a type constructor g. This is a contradiction with the hypothesis $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$. Hence, this case cannot occur.

- CASE I-HYP': We have $(\alpha_1 \geq \sigma_1) \in Q$, and σ_2 is α_1. By Lemma 2.5.6, we have $\sigma_1 \notin \mathcal{T}$, thus we have $\widehat{Q}(\mathsf{nf}(\sigma_2)) = \alpha_1 \in \mathcal{V}$, which is a contradiction (by hypothesis). Hence, this case cannot occur.

- CASE I-UP': We have $\sigma_1 = \forall\,(\alpha_1 {\geq} \forall\,(\alpha_2 {\diamond} \sigma'')\ \sigma')\ \sigma$ and $\sigma_2 = \forall\,(\alpha_2 {\diamond} \sigma'')\ \forall\,(\alpha_1 {\geq} \sigma')\ \sigma$. By hypothesis, we have $X \notin w(\sigma_1) - w(\sigma_2)$. By restrictions of Lemma 2.5.6, we have $\alpha_1 \in \mathsf{ftv}(\sigma)$, $\mathsf{nf}(\sigma) \neq \alpha_1$, $\alpha_2 \in \mathsf{ftv}(\sigma')$, $\sigma'' \notin \mathcal{T}$ $\mathbf{(\underline{5})}$, and $\mathsf{nf}(\sigma') \neq \alpha_2$. Let A be $X \star \diamond$. We have $w(\sigma_1) = w(\sigma) + X \times (w(\sigma') + A \times w_A(\sigma''))$. We have $w(\sigma_2) =$

$w(\sigma) + X \times w(\sigma') + A \times w_A(\sigma'')$. Hence, $w(\sigma_1) - w(\sigma_2) = A \times w_A(\sigma'') \times (X - 1)$. By Lemma 2.7.4 (page 92) and (5), $w_A(\sigma'') \neq 0$, thus $X \in w(\sigma_1) - w(\sigma_2)$, which is a contradiction (by hypothesis). Hence, this case cannot occur.

∘ Case I-Alias': We have $\sigma_1 = \forall\,(\alpha_1 \geq \sigma')\,\forall\,(\alpha_2 \geq \sigma')\;\sigma$ and $\sigma_2 = \forall\,(\alpha_1 \geq \sigma')\,\forall\,(\alpha_2 = \alpha_1)\;\sigma$. By restrictions of Lemma 2.5.6, we have α_1 and α_2 in $\mathsf{ftv}(\sigma)$, and $\sigma' \notin \mathcal{T}$ $\mathbf{(\underline{6})}$. By definition, we have $w(\sigma_1) = w(\sigma) + 2 \times X \times w(\sigma')$ and $w(\sigma_2) = w(\sigma) + X \times w(\sigma')$. Hence $w(\sigma_1) - w(\sigma_2) = X \times w(\sigma')$. By Lemma 2.7.4 (page 92) and (6), this implies $X \in w(\sigma_1) - w(\sigma_2)$, which is a contradiction. Hence, this case cannot occur.

∘ Case I-Rigid': We have $\sigma_1 = \forall\,(\alpha \geq \sigma')\;\sigma$ and $\sigma_2 = \forall\,(\alpha = \sigma')\;\sigma$. By restrictions of Lemma 2.3.1, we have σ' not in $\mathcal{T}$ $\mathbf{(\underline{7})}$, $\alpha \in \mathsf{ftv}(\sigma)$, and $\mathsf{nf}(\sigma) \neq \alpha$. Hence, $w(\sigma_1) = w(\sigma) + X \times w(\sigma')$ and $w(\sigma_2) = w(\sigma) + Y \times w_Y(\sigma')$. We get $w(\sigma_1) - w(\sigma_2) = Xw(\sigma') - Yw_Y(\sigma')$ $\mathbf{(\underline{8})}$. Since $X \notin Yw_Y(\sigma')$, the X-degree of $Yw_Y(\sigma')$ is 0. Since $w(\sigma')$ is not 0 by Lemma 2.7.4 (page 92) and (7), the X-degree of $Xw(\sigma')$ is not zero. Hence, $X \in Xw(\sigma') - Yw_Y(\sigma')$, that is, $X \in w(\sigma_1) - w(\sigma_2)$ by (8). This is a contradiction. Hence, this case cannot occur.

Conversely, we assume that $(Q)\;\sigma_1 \mathrel{-\mkern-11mu\sqsubseteq} \sigma_2$ holds. By Property 2.1.3.i, we have $\forall\,(\overline{Q})\;\sigma_1/ = \forall\,(Q)\;\sigma_2/$. By I-Abstract, we have $(Q)\;\sigma_1 \sqsubseteq \sigma_2$. It remains only to be shown that $X \notin w(\sigma_1) - w(\sigma_2)$ $\mathbf{(\underline{1})}$ holds. By Lemma 2.5.5 we have a derivation of $(Q)\;\sigma_1 \mathrel{-\mkern-11mu\sqsubseteq}^{\bar{\alpha}} \sigma_2$ $\mathbf{(\underline{2})}$. By Lemma 2.5.6, this derivation is restricted. We prove (1) by induction on the derivation of (2).

∘ Case A-Equiv': By Lemma 2.7.5 (page 93), we have $w_X(\sigma_1) - w_X(\sigma_2) = 0$.

∘ Case R-Trans: The premises are $(Q)\;\sigma_1 \mathrel{-\mkern-11mu\sqsubseteq}^{\bar{\alpha}} \sigma'_1$ $\mathbf{(\underline{3})}$ and $(Q)\;\sigma'_1 \mathrel{-\mkern-11mu\sqsubseteq}^{\bar{\alpha}} \sigma_2$ $\mathbf{(\underline{4})}$. If $\widehat{Q}(\mathsf{nf}(\sigma'_1))$ is in $\mathcal{V}$, then $(Q)\;\sigma_2 \equiv \sigma'_1$ holds by Lemma 2.1.6, thus $\widehat{Q}(\mathsf{nf}(\sigma_2))$ is in $\mathcal{V}$ by Lemma 1.5.9, which is a contradiction. Hence, $\widehat{Q}(\mathsf{nf}(\sigma'_1)) \notin \mathcal{V}$, and we get $X \notin w_X(\sigma_1) - w_X(\sigma'_1)$ by induction hypothesis on (3), as well as $X \notin w_X(\sigma'_1) - w_X(\sigma_2)$ by induction hypothesis on (4). This gives $X \notin w_X(\sigma_1) - w_X(\sigma_2)$ by addition.

∘ Case A-Hyp': We have $(\alpha = \sigma_1) \in Q$, and σ_2 is α. By Lemma 2.5.6, $\sigma_1 \notin \mathcal{T}$, that is, $\alpha \notin \mathsf{dom}(\widehat{Q})$. Hence, $\widehat{Q}(\sigma_2) = \widehat{Q}(\alpha) = \alpha$. This is a contradiction. This case is not possible.

∘ Case R-Context-R: We have $\sigma_1 = \forall\,(\alpha \diamond \sigma)\;\sigma'_1$ and $\sigma_2 = \forall\,(\alpha \diamond \sigma)\;\sigma'_2$. The premise is $(Q, \alpha \diamond \sigma)\;\sigma'_1 \mathrel{-\mkern-11mu\sqsubseteq}^{\bar{\alpha} \cup \{\alpha\}} \sigma'_2$ $\mathbf{(\underline{5})}$. We proceed by case analysis.

Subcase $\sigma \in \mathcal{T}$: Then $w(\sigma_1) = w(\sigma'_1)$ and $w(\sigma_2) = w(\sigma'_2)$. By induction hypothesis on (5), we get $X \notin w(\sigma'_1) - w(\sigma'_2)$, that is $X \notin w(\sigma_1) - w(\sigma_2)$.

In the following, we assume that $\sigma \notin \mathcal{T}$. By Lemmas 2.5.7 (page 83) and 2.1.4 (page 67), we have $\alpha \in \mathsf{ftv}(\sigma'_1)$ if and only if $\alpha \in \mathsf{ftv}(\sigma'_2)$ $\mathbf{(\underline{6})}$.

Subcase $\sigma'_2 \in \mathcal{V}$ and $\alpha \in \mathsf{ftv}(\sigma'_2)$: Necessarily $\mathsf{nf}(\sigma'_2) = \alpha$. Besides, we have $\alpha \in \mathsf{ftv}(\sigma'_1)$ from (6). Hence, $\sigma'_1 \equiv \alpha$ holds by Property 2.1.7.ii (page 68). Consequently, $\sigma_1 \equiv \sigma_2$ and $w(\sigma_1) - w(\sigma_2) = 0$.

Subcase $\sigma_2' \in \mathcal{V}$ and $\alpha \notin \mathsf{ftv}(\sigma_2')$: Then $\alpha \notin \mathsf{ftv}(\sigma_1')$ by (6). Hence, $w(\sigma_1) = w(\sigma_1')$ $\mathbf{(\underline{7})}$ and $w(\sigma_2) = w(\sigma_2')$ $\mathbf{(\underline{8})}$. Besides, we have $\sigma_2 \equiv \sigma_2'$ by Eq-Free. Hence, $\widehat{Q}(\mathsf{nf}(\sigma_2')) \approx \widehat{Q}(\mathsf{nf}(\sigma_2))$ holds by Lemma 1.5.9. This implies $\widehat{Q}(\mathsf{nf}(\sigma_2')) \notin \vartheta$. We conclude by induction hypothesis on (5), (7), and (8).

In the following, we assume that $\sigma_2' \notin \mathcal{V}$.

Subcase $\sigma_1' \equiv \alpha$: By Lemma 2.1.6 and (5), we get $(Q, \alpha \diamond \sigma)\ \sigma_1' \equiv \sigma_2'$, thus $(Q)\ \sigma_1 \equiv \sigma_2$ holds by R-Context-R, and $w(\sigma_1) - w(\sigma_2) = 0$ by Lemma 2.7.5 (page 93).

Otherwise let x be 1 if $\alpha \in \mathsf{ftv}(\sigma_1')$ and 0 otherwise. Let A be $X \star \diamond$. We have $w(\sigma_1) = w(\sigma_1') + x \times A \times w_A(\sigma)$ and $w(\sigma_2) = w(\sigma_2') + x \times A \times w_A(\sigma)$. Hence, $w(\sigma_1) - w(\sigma_2) = w(\sigma_1') - w(\sigma_2')$ and we conclude directly by induction hypothesis on (5).

◦ Case A-Context-L': We have $\sigma_1 = \forall\,(\alpha = \sigma_1')\ \sigma$ and $\sigma_2 = \forall\,(\alpha = \sigma_2')\ \sigma$. By Lemma 2.5.6, we have $\alpha \in \mathsf{ftv}(\sigma)$. By definition, we have $w(\sigma_1) = w_Y(\sigma_1') \times Y + w(\sigma)$ and $w(\sigma_2) = w_Y(\sigma_2') \times Y + w(\sigma)$. Hence, $w(\sigma_1) - w(\sigma_2) = (w_Y(\sigma_1') - w_Y(\sigma_2')) \times Y$, thus $X \notin w(\sigma_1) - w(\sigma_2)$.

◦ Case A-Alias': We have $\sigma_1 = \forall\,(\alpha_1 = \sigma)\ \forall\,(\alpha_2 = \sigma)\ \sigma'$, and α_1 and α_2 are in $\mathsf{ftv}(\sigma')$. Moreover, σ is not in $\mathcal{T}$. We have $w(\sigma_1) = w(\sigma') + 2 \times Y \times w_Y(\sigma)$ and $w(\sigma_2) = w(\sigma') + Y \times w_Y(\sigma)$. Hence, $w(\sigma_1) - w(\sigma_2) = Y \times w_Y(\sigma)$, thus $X \notin w(\sigma_1) - w(\sigma_2)$.

◦ Case A-Up': We have $\sigma_1 = \forall\,(\alpha_1 = \forall\,(\alpha_2 = \sigma'')\ \sigma')\ \sigma$ and $\sigma_2 = \forall\,(\alpha_2 = \sigma'')\ \forall\,(\alpha_1 = \sigma')\ \sigma$. Moreover, $\alpha_1 \in \mathsf{ftv}(\sigma)$, $\alpha_2 \in \mathsf{ftv}(\sigma')$, $\mathsf{nf}(\sigma)$ is not α_1, $\mathsf{nf}(\sigma')$ is not α_2, and σ'' is not in $\mathcal{T}$. We have $w(\sigma_1) = w(\sigma) + Y \times (w_Y(\sigma') + Y \times w_Y(\sigma''))$ and $w(\sigma_2) = w(\sigma) + Y \times w_Y(\sigma') + Y \times w_Y(\sigma'')$. Hence, $w(\sigma_1) - w(\sigma_2) = Y w_Y(\sigma'') \times (Y - 1)$. We see that $X \notin w(\sigma_1) - w(\sigma_2)$. ■

Proof of Property 2.7.9

Property i: It is shown by structural induction on σ.

◦ Case τ: We have $P(\sigma) = 0$ and $\#\sigma \geqslant n_\alpha$, thus the result holds.

◦ Case $\bot$: We have $P(\sigma) = 1$, $\#\sigma = 1$, and σ is closed, thus the result holds.

◦ Case $\forall\,(\alpha \diamond \sigma_1)\ \sigma_2$, with $\mathsf{nf}(\sigma_2) = \alpha$: By definition we have $P(\sigma) = P(\sigma_1)$, $n_\sigma = n_{\sigma_1}$, and $\#\sigma = \#\sigma_1$, thus we get the result by induction hypothesis on σ_1.

◦ Case $\forall\,(\alpha \diamond \sigma_1)\ \sigma_2$, with $\alpha \notin \mathsf{ftv}(\sigma_2)$: By definition we have $P(\sigma) = P(\sigma_2)$, $n_\sigma = n_{\sigma_2}$, and $\#\sigma = \#\sigma_2$, thus we get the result by induction hypothesis on σ_2.

◦ Case $\forall\,(\alpha \diamond \sigma_1)\ \sigma_2$ (other cases): By definition, $P(\sigma)$ is $P(\sigma_2) + X \times P(\sigma_1)$. By induction hypothesis, the coefficients of $P(\sigma_1)$ are bounded by $\#\sigma_1 - n_{\sigma_1}$, and the coefficients of $P(\sigma_2)$ are bounded by $\#\sigma_2 - n_{\sigma_2}$. Hence, the coefficients of $P(\sigma)$ are bounded by $\#\sigma_1 + \#\sigma_2 - n_{\sigma_1} - n_{\sigma_2}$. Since $\mathsf{ftv}(\sigma) = \mathsf{ftv}(\sigma_1) \cup \mathsf{ftv}(\sigma_2) - \{\alpha\}$, and $\alpha \in \mathsf{ftv}(\sigma_2)$, we have $n_\sigma \leqslant n_{\sigma_1} + n_{\sigma_2} - 1$. Consequently, $-n_{\sigma_1} - n_{\sigma_2} \leqslant -n_\sigma - 1$ holds. Hence, we have $\#\sigma_1 + \#\sigma_2 - n_{\sigma_1} - n_{\sigma_2} \leqslant \#\sigma_1 + \#\sigma_2 - n_\sigma - 1$, thus the coefficients of $P(\sigma)$ are bounded by $\#\sigma_1 + \#\sigma_2 - n_\sigma - 1$. Let k_α be the number of occurrences of α in σ_2 (that is, the cardinal of the set of all u such that $\sigma_2/u = \alpha$). We have $\#\sigma = \#\sigma_2 - k_\alpha + k_\alpha \times \#\sigma_1$. Hence, $\#\sigma = \#\sigma_2 + k_\alpha \times (\#\sigma_1 - 1)$. Since $\alpha \in \mathsf{ftv}(\sigma_2)$, we have $k_\alpha \geqslant 1$, thus $\#\sigma \geqslant \#\sigma_2 + \#\sigma_1 - 1$ **(1)**, that is $\#\sigma_1 + \#\sigma_2 \leqslant \#\sigma + 1$. Since all the coefficients of $P(\sigma)$ are bounded by $\#\sigma_1 + \#\sigma_2 - n_\sigma - 1$, they are also bounded by $\#\sigma - n_\sigma$, which is the expected result.

Property ii: It is shown by structural induction on σ. All cases are straightforward, except the last one:

◦ Case $\forall\,(\alpha \diamond \sigma_1)\ \sigma_2$, when $\sigma_2 \not\equiv \alpha$ and $\alpha \in \mathsf{ftv}(\sigma_2)$. By definition, $P(\sigma) = P(\sigma_2) + X \times P(\sigma_1)$, thus $d(\sigma) = \max(d(\sigma_2), d(\sigma_1) + 1)$. We have $d(\sigma_2) \leqslant \#\sigma_2$ by induction hypothesis as well as $d(\sigma_1) \leqslant \#\sigma_1$. Hence, $d(\sigma) \leqslant \max(\#\sigma_2, \#\sigma_1 + 1)$ **(2)**. Moreover, as shown above (1), we have $\#\sigma \geqslant \#\sigma_2 + \#\sigma_1 - 1$ **(3)**. Additionally, we have $\#\sigma_2 \leqslant \#\sigma_2 + \#\sigma_1 - 1$ **(4)**. Since $\sigma_2 \not\equiv \alpha$, and $\alpha \in \mathsf{ftv}(\sigma_2)$, we must have $\#\sigma_2 \geqslant 2$ by Property 2.1.5.ii (page 67). Hence, $\#\sigma_1 + 1 \leqslant \#\sigma_1 + \#\sigma_2 - 1$ **(5)**. As a consequence of (4) and (5), we have $\max(\#\sigma_2, \#\sigma_1 + 1) \leqslant \#\sigma_2 + \#\sigma_1 - 1$, thus $\max(\#\sigma_2, \#\sigma_1 + 1) \leqslant \#\sigma$ holds by (3). Finally, we get $d(\sigma) \leqslant \#\sigma$ by (2). ■

Proof of Property 2.8.1

Property i: Intuitively, this proof is quite long because we have to consider many critical pairs. Two rules can be used to derive $(Q)\ \sigma_1 \dot{\sqsubseteq}^{\bar{\alpha}}\ \sigma_2$, namely StSh-Up and StSh-Alias (note that the prefix is unconstrained). Similarly, two rules can be used to derive $(Q)\ \sigma_1 \dot{\sqsubseteq}^{\bar{\alpha}}\ \sigma_3$. By symmetry, this makes three cases to consider:

◦ Case StSh-Up and StSh-Up: We have

$$\sigma_1 = C_r(\forall\,(\beta = \forall\,(Q_1)\ \forall\,(\alpha = \sigma_a)\ \sigma_b)\ \sigma_c)\ \mathbf{(\underline{1})} \qquad \sigma_2 = C_r(\forall\,(\alpha = \sigma_a)\ \forall\,(\beta = \forall\,(Q_1)\ \sigma_b)\ \sigma_c)$$

$$\sigma_1 = C'_r(\forall\,(\beta' = \forall\,(Q_2)\ \forall\,(\alpha' = \sigma'_a)\ \sigma'_b)\ \sigma'_c)\ \mathbf{(\underline{2})}$$

$$\sigma_3 = C'_r(\forall\,(\alpha' = \sigma'_a)\ \forall\,(\beta' = \forall\,(Q_2)\ \sigma'_b)\ \sigma'_c)$$

Besides, the premises give the following hypotheses **(3)**:

$$\beta \in \mathsf{ftv}(\sigma_c) \qquad \beta' \in \mathsf{ftv}(\sigma'_c) \qquad \alpha \in \mathsf{ftv}(\sigma_b) \qquad \alpha' \in \mathsf{ftv}(\sigma'_b) \qquad \sigma_b, \sigma'_b, \sigma_c, \sigma'_c \notin \mathcal{V}$$

$$\sigma_a, \sigma'_a \notin \mathcal{T} \qquad \alpha \notin \mathsf{dom}(Q\overline{C_r}Q_1) \qquad \alpha' \notin \mathsf{dom}(Q\overline{C_r}Q_2)$$

We proceed by case analysis on C_r and C'_r.

SUBCASE $C_r = C'_r$: Then from (1) and (2), we get $\forall\,(Q_2)\,\forall\,(\alpha'\!=\!\sigma'_a)\;\sigma'_b = \forall\,(Q_1)\,\forall\,(\alpha=\sigma_a)\;\sigma_b$. If α is α', we have $\sigma_3 = \sigma_2$, thus we get the result by taking $\sigma_4 = \sigma_2$. Otherwise, with no loss of generality, we can freely assume that $\alpha \in \mathsf{dom}(Q_2)$. Hence, we have σ_1 of the form $C_r(\forall\,(\beta = \forall\,(Q_a, \alpha = \sigma_a, Q_b, \alpha' = \sigma'_a)\;\sigma'_b)\;\sigma_c)$, σ_2 of the form $C_r(\forall\,(\alpha=\sigma_a)\,\forall\,(\beta=\forall\,(Q_aQ_b,\alpha'=\sigma'_a)\;\sigma'_b)\;\sigma_c$, and σ_3 of the form $C_r(\forall\,(\alpha'=\sigma'_a)\,\forall\,(\beta=\forall\,(Q_a,\alpha{=}\sigma_a,Q_b)\;\sigma'_b)\;\sigma_c$. Then taking $\sigma_4 = C_r(\forall\,(\alpha{=}\sigma_a,\alpha'{=}\sigma'_a)\,\forall\,(\beta{=}\forall\,(Q_aQ_b)\;\sigma'_b)\;\sigma_c)$, we can derive $(Q)\;\sigma_2 \mathrel{\dot{\sqsubseteq}^{\bar\alpha}} \sigma_4$ **($\underline{4}$)** and $(Q)\;\sigma_3 \mathrel{\dot{\sqsubseteq}^{\bar\alpha}} \sigma_4$ **($\underline{5}$)** by STSH-UP. The premises of (4) and (5) are ensured by (3).

SUBCASE C_r and C'_r disjoint: By Lemma 2.4.6 (page 76), there exists a two-hole context C^2 such that $C_r = C^2([\ ], \forall\,(\beta' = \forall\,(Q_2)\,\forall\,(\alpha' = \sigma'_a)\;\sigma'_b)\;\sigma'_c)$ and $C'_r = C^2(\forall\,(\beta=\forall\,(Q_1)\,\forall\,(\alpha=\sigma_a)\;\sigma_b)\;\sigma_c,[\ ])$. Then taking $\sigma_4 = C^2(\forall\,(\alpha=\sigma_a)\,\forall\,(\beta=\forall\,(Q_1)\;\sigma_b)\;\sigma_c, \forall\,(\alpha'=\sigma'_a)\,\forall\,(\beta'=\forall\,(Q_2)\;\sigma'_b)\;\sigma'_c)$ gives the expected result.

SUBCASE C_r and C'_r nested: without loss of generality, we can freely assume that C'_r is of the form $C_r(C)$. Hence, we have

$$\sigma_1 = C_r(\forall\,(\beta{=}\forall\,(Q_1)\,\forall\,(\alpha{=}\sigma_a)\;\sigma_b)\;\sigma_c)\ \mathbf{(\underline{6})} \qquad \sigma_2 = C_r(\forall\,(\alpha{=}\sigma_a)\,\forall\,(\beta{=}\forall\,(Q_1)\;\sigma_b)\;\sigma_c)$$

$$\sigma_1 = C_r(C(\forall\,(\beta'=\forall\,(Q_2)\,\forall\,(\alpha'=\sigma'_a)\;\sigma'_b)\;\sigma'_c))\ \mathbf{(\underline{7})}$$

$$\sigma_3 = C_r(C(\forall\,(\alpha'=\sigma'_a)\,\forall\,(\beta'=\forall\,(Q_2)\;\sigma'_b)\;\sigma'_c))$$

From (6) and (7), we get $C(\forall\,(\beta'{=}\forall\,(Q_2)\,\forall\,(\alpha'{=}\sigma'_a)\;\sigma'_b)\;\sigma'_c) = \forall\,(\beta{=}\forall\,(Q_1)\,\forall\,(\alpha{=}\sigma_a)\;\sigma_b)\;\sigma_c$. We have six choices for C:

- C is of the form $\forall\,(\beta{=}\forall\,(Q_1)\,\forall\,(\alpha{=}\sigma_a)\;\sigma_b)\;C'$. We get the expected solution by taking $\sigma_4 = C_r(\forall\,(\alpha=\sigma_a)\,\forall\,(\beta=\forall\,(Q_1)\;\sigma_b)\;C'(\forall\,(\alpha'=\sigma'_a)\,\forall\,(\beta'=\forall\,(Q_2)\;\sigma'_b)\;\sigma'_c))$

- C is of the form $\forall\,(\beta = \forall\,(Q')\;[\ ])\;\sigma_c$ and $\mathsf{dom}(Q') \subset \mathsf{dom}(Q_1)$: Then σ'_c is $\forall\,(Q'')\,\forall\,(\alpha=\sigma_a)\;\sigma_b$. We get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha=\sigma_a)\,\forall\,(\beta=\forall\,(Q')\,\forall\,(\alpha'=\sigma'_a)\,\forall\,(\beta'=\forall\,(Q_2)\;\sigma'_b)\,\forall\,(Q'')\;\sigma_b)\;\sigma_c)$.

- C is of the form $\forall\,(\beta=\forall\,(Q_1)\;[\ ])\;\sigma_c$: Then β' is α. We get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha'=\sigma'_a)\,\forall\,(\alpha=\forall\,(Q_2)\;\sigma'_b)\,\forall\,(\beta=\forall\,(Q_1)\;\sigma'_c)\;\sigma_c)$.

- C is of the form $\forall\,(\beta=\forall\,(Q_a)\,\forall\,(\gamma=C')\,\forall\,(Q_b)\,\forall\,(\alpha=\sigma_a)\;\sigma_b)\;\sigma_c$: We get the expected solution by taking $\sigma_4 = C_r(\forall\,(\alpha{=}\sigma_a)\,\forall\,(\beta{=}\forall\,(Q_a)\,\forall\,(\gamma{=}C'(\forall\,(\alpha'{=}\sigma'_a)\,\forall\,(\beta'=\forall\,(Q_2)\;\sigma'_b)\;\sigma'_c))\,\forall\,(Q_b)\;\sigma_b)\;\sigma_c)$

- C is of the form $\forall\,(\beta{=}\forall\,(Q_1)\,\forall\,(\alpha{=}C')\;\sigma_b)\;\sigma_c$: We get the expected solution by taking $\sigma_4 = C_r(\forall\,(\alpha=C'(\forall\,(\alpha'=\sigma'_a)\,\forall\,(\beta'=\forall\,(Q_2)\;\sigma'_b)\;\sigma'_c))\,\forall\,(\beta=\forall\,(Q_1)\;\sigma_b)\;\sigma_c)$

- C is of the form $\forall\,(\beta = \forall\,(Q_1)\;\forall\,(\alpha = \sigma_a)\;C')\;\sigma_c$: We get the expected solution by taking $\sigma_4 = C_r(\forall\,(\alpha = \sigma_a)\;\forall\,(\beta = \forall\,(Q_1)\;C'(\forall\,(\alpha' = \sigma'_a)\;\forall\,(\beta' = \forall\,(Q_2)\;\sigma'_b)\;\sigma'_c))\;\sigma_c)$

◦ Case StSh-Up and StSh-Alias: We have

$$\begin{array}{ll}
\sigma_1 = C_r(\forall\,(\beta = \forall\,(Q_1)\;\forall\,(\alpha = \sigma_a)\;\sigma_b)\;\sigma_c) & \sigma_1 = C'_r(\forall\,(\alpha_1 = \sigma'_1)\;\forall\,(Q')\;\forall\,(\alpha_2 = \sigma'_2)\;\sigma''_1) \\
\sigma_2 = C_r(\forall\,(\alpha = \sigma_a)\;\forall\,(\beta = \forall\,(Q_1)\;\sigma_b)\;\sigma_c) & \sigma_3 = C'_r(\forall\,(\alpha_1 = \sigma'_1)\;\forall\,(Q')\;\forall\,(\alpha_2 = \alpha_1)\;\sigma''_1)
\end{array}$$

and by hypothesis, $\sigma'_1 \equiv \sigma'_2$. We proceed by case analysis on C_r and C'_r.

Subcase $C_r = C'_r$: then, $\beta = \alpha_1$, thus we have

$$\begin{array}{cc}
\sigma_1 = C_r(\forall\,(\beta = \forall\,(Q_1)\;\forall\,(\alpha = \sigma_a)\;\sigma_b)\;\sigma_c) & \sigma_1 = C_r(\forall\,(\beta = \sigma'_1)\;\forall\,(Q')\;\forall\,(\alpha_2 = \sigma'_2)\;\sigma''_1) \\
\sigma_2 = C_r(\forall\,(\alpha = \sigma_a)\;\forall\,(\beta = \forall\,(Q_1)\;\sigma_b)\;\sigma_c) & \sigma_3 = C_r(\forall\,(\beta = \sigma'_1)\;\forall\,(Q')\;\forall\,(\alpha_2 = \beta)\;\sigma''_1) \\
\sigma'_1 = \forall\,(Q_1)\;\forall\,(\alpha = \sigma_a)\;\sigma_b & \sigma_c = \forall\,(Q')\;\forall\,(\alpha_2 = \sigma'_2)\;\sigma''_1
\end{array}$$

Taking $\sigma_4 = C_r(\forall\,(\alpha = \sigma_a)\;\forall\,(\beta = \forall\,(Q_1)\;\sigma_b)\;\forall\,(Q')\;\forall\,(\alpha_2 = \beta)\;\sigma''_1)$ is appropriate.

Subcase C_r and C'_r are disjoint: by Lemma 2.4.6 (page 76), there exists a two-hole context C^2 such that $C_r = C^2([\,], \forall\,(\alpha_1 = \sigma'_1)\;\forall\,(Q')\;\forall\,(\alpha_2 = \sigma'_2)\;\sigma''_1)$ and $C'_r = C^2(\forall\,(\beta = \forall\,(Q_1)\;\forall\,(\alpha = \sigma_a)\;\sigma_b)\;\sigma_c, [\,])$. Then taking $\sigma_4 = C^2(\forall\,(\alpha = \sigma_a)\;\forall\,(\beta = \forall\,(Q_1)\;\sigma_b)\;\sigma_c, \forall\,(\alpha_1 = \sigma'_1)\;\forall\,(Q')\;\forall\,(\alpha_2 = \alpha_1)\;\sigma''_1)$ is appropriate.

Subcase C_r is of the form $C'_r(C)$: we have

$$\sigma_1 = C'_r(C(\forall\,(\beta = \forall\,(Q_1)\;\forall\,(\alpha = \sigma_a)\;\sigma_b)\;\sigma_c))$$

$$\sigma_1 = C'_r(\forall\,(\alpha_1 = \sigma'_1)\;\forall\,(Q')\;\forall\,(\alpha_2 = \sigma'_2)\;\sigma''_1)$$

$$\sigma_2 = C'_r(C(\forall\,(\alpha = \sigma_a)\;\forall\,(\beta = \forall\,(Q_1)\;\sigma_b)\;\sigma_c))$$

$$\sigma_3 = C'_r(\forall\,(\alpha_1 = \sigma'_1)\;\forall\,(Q')\;\forall\,(\alpha_2 = \alpha_1)\;\sigma''_1)$$

We have five choices for C:

- $C = \forall\,(\alpha_1 = C')\;\forall\,(Q')\;\forall\,(\alpha_2 = \sigma'_2)\;\sigma''_1$: we get the expected solution by taking $\sigma_4 = C'_r(\forall\,(\alpha_1 = C'(\forall\,(\alpha = \sigma_a)\;\forall\,(\beta = \forall\,(Q_1)\;\sigma_b)\;\sigma_c))\;\forall\,(Q')\;\forall\,(\alpha_2 = \alpha_1)\;\sigma''_1)$
- $C = \forall\,(\alpha_1 = \sigma'_1)\;\forall\,(Q_a)\;[\,]$ with $\alpha_2 \notin \mathsf{dom}(Q_a)$. We have $\forall\,(Q_a)\;\forall\,(\beta = \forall\,(Q_1)\;\forall\,(\alpha = \sigma_a)\;\sigma_b)\;\sigma_c = \forall\,(Q')\;\forall\,(\alpha_2 = \sigma'_2)\;\sigma''_1$.

If $\beta = \alpha_2$, then taking $\sigma_4 = C'_r(\forall\,(\alpha\!=\!\sigma_a)\ \forall\,(\alpha_1\!=\!\forall\,(Q_1)\ \sigma_b)\ \forall\,(Q')\ \forall\,(\alpha_2\!=\!\alpha_1)\ \sigma_c)$ is appropriate.

Otherwise, $\beta \in \mathsf{dom}(Q')$ and σ_c is of the form $\forall\,(Q_2)\ \forall\,(\alpha_2\!=\!\sigma'_2)\ \sigma''_1$. Then taking $\sigma_4 = C'_r(\forall\,(\alpha_1 = \sigma'_1)\ \forall\,(Q_a)\ \forall\,(\alpha = \sigma_a)\ \forall\,(\beta = \forall\,(Q_1)\ \sigma_b)\ \forall\,(Q_2)\ \forall\,(\alpha_2 = \alpha_1)\ \sigma''_1)$ is appropriate.

- $C = \forall\,(\alpha_1 = \sigma'_1)\ \forall\,(Q_a)\ \forall\,(\gamma = C')\ \forall\,(Q_b)\ \forall\,(\alpha_2 = \sigma'_2)\ \sigma''_1$: we get the expected solution by taking $\sigma_4 = C'_r(\forall\,(\alpha_1 = \sigma'_1)\ \forall\,(Q_a)\ \forall\,(\gamma = C'(\forall\,(\alpha = \sigma_a)\ \forall\,(\beta = \forall\,(Q_1)\ \sigma_b)\ \sigma_c))\ \forall\,(Q_b)\ \forall\,(\alpha_2 = \alpha_1)\ \sigma''_1)$
- $C = \forall\,(\alpha_1 = \sigma'_1)\ \forall\,(Q')\ \forall\,(\alpha_2 = C')\ \sigma''_1$: we get the expected solution by taking $\sigma_4 = C'_r(\forall\,(\alpha_1 = C'(\forall\,(\alpha = \sigma_a)\ \forall\,(\beta = \forall\,(Q_1)\ \sigma_b)\ \sigma_c))\ \forall\,(Q')\ \forall\,(\alpha_2 = \alpha_1)\ \sigma''_1)$
- $C = \forall\,(\alpha_1 = \sigma'_1)\ \forall\,(Q')\ \forall\,(\alpha_2 = \sigma'_2)\ C'$: we get the expected solution by taking $\sigma_4 = C'_r(\forall\,(\alpha_1 = \sigma'_1)\ \forall\,(Q')\ \forall\,(\alpha_2 = \alpha_1)\ C'(\forall\,(\alpha = \sigma_a)\ \forall\,(\beta = \forall\,(Q_1)\ \sigma_b)\ \sigma_c))$

Subcase C'_r is of the form $C_r(C)$: we have

$$\sigma_1 = C_r(\forall\,(\beta = \forall\,(Q_1)\ \forall\,(\alpha = \sigma_a)\ \sigma_b)\ \sigma_c)\ (\underline{\mathbf{8}})$$

$$\sigma_1 = C_r(C(\forall\,(\alpha_1 = \sigma'_1)\ \forall\,(Q')\ \forall\,(\alpha_2 = \sigma'_2)\ \sigma''_1))\ (\underline{\mathbf{9}})$$

$$\sigma_2 = C_r(\forall\,(\alpha = \sigma_a)\ \forall\,(\beta = \forall\,(Q_1)\ \sigma_b)\ \sigma_c)$$

$$\sigma_3 = C_r(C(\forall\,(\alpha_1 = \sigma'_1)\ \forall\,(Q')\ \forall\,(\alpha_2 = \alpha_1)\ \sigma''_1))$$

From (8) and (9), we get

$$C(\forall\,(\alpha_1 = \sigma'_1)\ \forall\,(Q')\ \forall\,(\alpha_2 = \sigma'_1)\ \sigma''_1)) = \forall\,(\beta = \forall\,(Q_1)\ \forall\,(\alpha = \sigma_a)\ \sigma_b)\ \sigma_c$$

We have eight choices for C:

- C is of the form $\forall\,(\beta = \forall\,(Q_1)\ \forall\,(\alpha = \sigma_a)\ \sigma_b)\ C'$: we get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha\!=\!\sigma_a)\ \forall\,(\beta\!=\!\forall\,(Q_1)\ \sigma_b)\ C'(\forall\,(\alpha_1\!=\!\sigma'_1)\ \forall\,(Q')\ \forall\,(\alpha_2\!=\!\alpha_1)\ \sigma''_1))$.
- C is of the form $\forall\,(\beta\!=\!\forall\,(Q_a)\ [\,])\ \sigma_c$ with $\mathsf{dom}(Q_a) \subset \mathsf{dom}(Q_1)$ and $\alpha = \alpha_2$: then, $Q_1 = (Q_a, \alpha_1 = \sigma'_1, Q')$. We get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha_1 = \sigma'_1, \alpha = \alpha_1)\ \forall\,(\beta = \forall\,(Q_a, Q')\ \sigma_b)\ \sigma_c)$.
- C is of the form $\forall\,(\beta\!=\!\forall\,(Q_a)\ [\,])\ \sigma_c$ with $\mathsf{dom}(Q_a) \subset \mathsf{dom}(Q_1)$ and $\alpha \in \mathsf{dom}(Q')$: then, $Q' = (Q_b, \alpha = \sigma_a, Q_c)$ We get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha = \sigma_a)\ \forall\,(\beta = \forall\,(Q_a, \alpha_1 = \sigma'_1, Q_b, Q_c, \alpha_2 = \alpha_1)\ \sigma''_1)))$.

- C is of the form $\forall\,(\beta{=}\forall\,(Q_a)\;[\,])\;\sigma_c$ with $\mathsf{dom}(Q_a) \subset \mathsf{dom}(Q_1)$ and $\alpha_2 \in \mathsf{dom}(Q_1)$: then, $Q_1 = (Q_a, \alpha_1 = \sigma'_1, Q', \alpha_2 = \sigma'_2, Q_b)$ We get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha = \sigma_a)\;\forall\,(\beta = \forall\,(Q_a, \alpha_1 = \sigma'_1, Q', \alpha_2 = \alpha_1, Q_b)\;\sigma_b)\;\sigma_c)$.
- C is of the form $\forall\,(\beta = \forall\,(Q_1)\;[\,])\;\sigma_c$ (that is, $\alpha_1 = \alpha$): we get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha_1 = \sigma'_1, \alpha_2 = \alpha_1)\;\forall\,(\beta = \forall\,(Q_1, Q')\;\sigma''_1)\;\sigma_c)$.
- C is of the form $\forall\,(\beta = \forall\,(Q_a)\;\forall\,(\gamma = C')\;\forall\,(Q_b)\;\forall\,(\alpha = \sigma_a)\;\sigma_b)\;\sigma_c$: we get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha{=}\sigma_a)\;\forall\,(\beta{=}\forall\,(Q_a, \gamma{=}C'(\forall\,(\alpha_1{=}\sigma'_1, Q', \alpha_2{=}\alpha_1)\;\sigma''_1), Q_b)\;\sigma_b)\;\sigma_c)$.
- C is of the form $\forall\,(\beta = \forall\,(Q_1)\;\forall\,(\alpha = C')\;\sigma_b)\;\sigma_c$: we get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha = C'(\forall\,(\alpha_1 = \sigma'_1, Q', \alpha_2 = \alpha_1)\;\sigma''_1))\;\forall\,(\beta = \forall\,(Q_1)\;\sigma_b)\;\sigma_c)$.
- C is of the form $\forall\,(\beta = \forall\,(Q_1)\;\forall\,(\alpha = \sigma_a)\;C')\;\sigma_c$: we get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha = \sigma_a)\;\forall\,(\beta = \forall\,(Q_1)\;C'(\forall\,(\alpha_1 = \sigma'_1, Q', \alpha_2 = \alpha_1)\;\sigma''_1))\;\sigma_c)$.

○ Case StSh-Alias and StSh-Alias: we have

$$\sigma_1 = C_r(\forall\,(\alpha_1 = \sigma_a, Q, \alpha_2 = \sigma_b)\;\sigma) \qquad \sigma_2 = C_r(\forall\,(\alpha_1 = \sigma_a, Q, \alpha_2 = \alpha_1)\;\sigma)$$

$$\sigma_1 = C'_r(\forall\,(\alpha'_1 = \sigma'_a, Q', \alpha'_2 = \sigma'_b)\;\sigma') \qquad \sigma_3 = C'_r(\forall\,(\alpha'_1 = \sigma'_a, Q', \alpha'_2 = \alpha'_1)\;\sigma')$$

By hypothesis, we have $\sigma_a \equiv \sigma_b$ and $\sigma'_a \equiv \sigma'_b$.

We proceed by case analysis on C_r and C'_r.

Subcase $C_r = C'_r$: then, $\alpha_1 = \alpha'_1$. If α_2 is α'_2, then $\sigma_2 = \sigma_3$ and we get the result by taking $\sigma_4 = \sigma_3$. Otherwise, without loss of generality, we can assume that σ_1 is of the form $C_r(\forall\,(\alpha_1 = \sigma_a, Q, \alpha_2 = \sigma_b, Q_1, \alpha'_2 = \sigma_1)\;\sigma')$ We get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha_1 = \sigma_a, Q, \alpha_2 = \alpha_1, Q_1, \alpha'_2 = \alpha_1)\;\sigma')$

Subcase C_r and C'_r are disjoint: by Lemma 2.4.6 (page 76), there exists a two-hole context C^2 such that $C_r = C^2([\,], \forall\,(\alpha'_1 = \sigma'_a, Q, \alpha'_2 = \sigma'_b)\;\sigma')$ and $C'_r = C^2(\forall\,(\alpha_1 = \sigma_a, Q, \alpha_2 = \sigma_b)\;\sigma, [\,])$. Then we get the expected result by taking $\sigma_4 = C^2(\forall\,(\alpha_1 = \sigma_a, Q, \alpha_2 = \alpha_1)\;\sigma, \forall\,(\alpha'_1 = \sigma'_a, Q, \alpha'_2 = \alpha'_1)\;\sigma')$.

Subcase C_r and C'_r are nested: without loss of generality, we can freely assume that C'_r is of the form $C_r(C)$. Hence, we have

$$\sigma_1 = C_r(\forall\,(\alpha_1 = \sigma_a, Q, \alpha_2 = \sigma_b)\;\sigma)\;(\underline{\mathbf{10}}) \qquad \sigma_2 = C_r(\forall\,(\alpha_1 = \sigma_a, Q, \alpha_2 = \alpha_1)\;\sigma)$$

$$\sigma_1 = C_r(C(\forall\,(\alpha'_1{=}\sigma'_a, Q', \alpha'_2{=}\sigma'_b)\;\sigma'))\;(\underline{\mathbf{11}}) \qquad \sigma_3 = C_r(C(\forall\,(\alpha'_1{=}\sigma'_a, Q', \alpha'_2{=}\alpha'_1)\;\sigma'))$$

From (10) and (11), we get

$$\forall\,(\alpha_1 = \sigma_a, Q, \alpha_2 = \sigma_b)\ \sigma = C(\forall\,(\alpha'_1 = \sigma'_a, Q', \alpha'_2 = \sigma'_b)\ \sigma')$$

We have eight choices for C:

- $C = \forall\,(\alpha_1 = C', Q, \alpha_2 = \sigma_b)\ \sigma$: Then we get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha_1 = C'(\forall\,(\alpha'_1 = \sigma'_a, Q', \alpha'_2 = \alpha'_1)\ \sigma'), Q, \alpha_2 = \alpha_1)\ \sigma)$.
- $C = \forall\,(\alpha_1{=}\sigma_a, Q_1, \forall\,(\gamma{=}C'), Q_2, \alpha_2{=}\sigma_b)\ \sigma$: Then we get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha_1{=}\sigma_a, Q_1, \forall\,(\gamma{=}C'(\forall\,(\alpha'_1{=}\sigma'_a, Q', \alpha'_2{=}\alpha'_1)\ \sigma')), Q_2, \alpha_2{=}\alpha_1)\ \sigma$.
- $C = \forall\,(\alpha_1 = \sigma_a, Q, \alpha_2 = C')\ \sigma$: Then we get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha_1 = C'(\forall\,(\alpha'_1 = \sigma'_a, Q', \alpha'_2 = \alpha'_1)\ \sigma')Q, \alpha_2 = \alpha_1)\ \sigma$.
- $C = \forall\,(\alpha_1 = \sigma_a, Q, \alpha_2 = \sigma_b)\ C'$: Then we get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha_1 = \sigma_a, Q, \alpha_2 = \alpha_1)\ C'(\forall\,(\alpha'_1 = \sigma'_a, Q', \alpha'_2 = \alpha'_1)\ \sigma'))$.
- $C = \forall\,(\alpha_1 = \sigma_a, Q_1)\ [\ ]$ with $\mathsf{dom}(Q_1) \subset \mathsf{dom}(Q)$ and $\alpha'_2 \in \mathsf{dom}(Q)$: We have $Q = (Q_1, \alpha'_1 = \sigma'_a, Q', \alpha'_2 = \sigma'_b, Q_2)$. Then we get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha_1 = \sigma_a, Q_1, \alpha'_1 = \sigma'_a, Q', \alpha'_2 = \alpha'_1, Q_2, \alpha_2 = \alpha_1)\ \sigma)$.
- $C = \forall\,(\alpha_1 = \sigma_a, Q_1)\ [\ ]$ with $\mathsf{dom}(Q_1) \subset \mathsf{dom}(Q)$ and $\alpha_2 = \alpha'_2$: we have $\sigma_1 = \sigma'_1$ and $Q = (Q_1, \alpha'_1 = \sigma'_a, Q')$. Then we get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha_1 = \sigma_a, Q_1, \alpha'_1 = \alpha_1, Q', \alpha_2 = \alpha_1)\ \sigma)$.
- $C = \forall\,(\alpha_1 = \sigma_a, Q_1)\ [\ ]$ with $\mathsf{dom}(Q_1) \subset \mathsf{dom}(Q)$ and $\alpha_2 \in \mathsf{dom}(Q')$: we have $Q' = (Q_a, \alpha_2{=}\sigma_b, Q_b)$. Then we get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha_1{=}\sigma_a, Q_1, \alpha'_1 = \sigma'_a, Q_a, \alpha_2 = \alpha_1, Q_b, \alpha'_2 = \alpha'_1)\ \sigma')$.
- $C = \forall\,(\alpha_1 = \sigma_a, Q)\ [\ ]$ (that is, $\alpha_2 = \alpha'_1$): we have $\sigma_1 = \sigma'_1$. Then we get the expected result by taking $\sigma_4 = C_r(\forall\,(\alpha_1 = \sigma_a, Q, \alpha'_1 = \alpha_1, Q', \alpha'_2 = \alpha_1)\ \sigma')$.

<u>Property ii</u>: By hypothesis, we have σ_1, σ_2 and σ_3 such that $(Q)\ \sigma_1\ (\equiv\dot{\sqsubseteq}^{\emptyset})\ \sigma_2$ and $(Q)\ \sigma_2\ (\equiv\dot{\sqsubseteq}^{\emptyset})\ \sigma_3$ hold. We have to show that there exists σ_4 such that $(Q)\ \sigma_2\ (\equiv\dot{\sqsubseteq}^{\emptyset})^*\ \sigma_4$ and $(Q)\ \sigma_3\ (\equiv\dot{\sqsubseteq}^{\emptyset})^*\ \sigma_4$. By Property 2.6.2.i (page 85), the relations $(\equiv\dot{\sqsubseteq}^{\emptyset})^*$ and $\sqsubseteq$ are equivalent, thus it suffices to show that there exists σ_4 such that $(Q)\ \sigma_2 \sqsubseteq \sigma_4$ and $(Q)\ \sigma_3 \sqsubseteq \sigma_4$. If $(Q)\ \sigma_1 \equiv \sigma_2$ or $(Q)\ \sigma_1 \equiv \sigma_3$ hold, then taking (respectively) $\sigma_4 = \sigma_3$ or $\sigma_4 = \sigma_2$ gives the expected result. Otherwise, by definition, there exist σ_1^a, σ'_2, σ_1^b and σ'_3 such that we have

$$(Q)\ \sigma_1 \equiv \sigma_1^a\ (\underline{\mathbf{1}}) \qquad (Q)\ \sigma_1^a\ \dot{\sqsubseteq}^{\emptyset}\ \sigma'_2\ (\underline{\mathbf{2}}) \qquad (Q)\ \sigma'_2 \equiv \sigma_2 \qquad (Q)\ \sigma_1 \equiv \sigma_1^b$$

$$(Q)\ \sigma_1^b\ \dot{\sqsubseteq}^{\emptyset}\ \sigma'_3 \qquad (Q)\ \sigma'_3 \equiv \sigma_3$$

Figure A.1: Commutation

$$\begin{tikzcd}
\sigma_1 \arrow[r, "\equiv"] \arrow[d, "\equiv"'] & \sigma_1^a \arrow[r, "\dot{\sqsubseteq}^{\emptyset}"] \arrow[d, "\equiv"'] & \sigma'_2 \arrow[r, "\equiv"] \arrow[d, dotted, "\equiv"] & \sigma_2 \\
\sigma_1^b \arrow[r, "\equiv"] \arrow[d, "\dot{\sqsubseteq}^{\emptyset}"'] & \sigma'_1 \arrow[r, dotted, "\dot{\sqsubseteq}^{\emptyset}"] \arrow[d, dotted, "\dot{\sqsubseteq}^{\emptyset}"'] & \sigma''_2 \arrow[d, dotted, "\sqsubseteq"] & \\
\sigma'_3 \arrow[r, dotted, "\equiv"'] \arrow[d, "\equiv"'] & \sigma''_3 \arrow[r, dotted, "\sqsubseteq"'] & \sigma_4 & \\
\sigma_3 & & &
\end{tikzcd}$$

Let σ'_1 be $\mathsf{nf}(\sigma_1)$. By Property 1.5.6.iv (page 51), σ'_1 is in normal form, and by Property 1.5.6.i (page 51), $\sigma'_1 \equiv \sigma_1$ holds.

We represent these relations by the solid arrows of figure A.1, to be read under prefix Q. The dotted arrows correspond to relations shown below.

By Property 1.5.11.i (page 54) and (1), $\mathsf{nf}(\sigma_1^a) \approx \sigma'_1$ holds. By (2), Properties 2.6.3.i (page 86) and 2.6.3.ii (page 86), there exists σ''_2 such that $(Q)\ \sigma'_2 \equiv \sigma''_2$ and $(Q)\ \sigma'_1 \dot{\sqsubseteq}^{\emptyset} \sigma''_2$. Similarly, there exists σ''_3 such that $(Q)\ \sigma'_3 \equiv \sigma''_3$ and $(Q)\ \sigma'_1 \dot{\sqsubseteq}^{\emptyset} \sigma''_3$. Hence, we have

$$(Q)\ \sigma_1 \equiv \sigma'_1 \qquad (Q)\ \sigma'_1 \dot{\sqsubseteq}^{\emptyset} \sigma''_2 \qquad (Q)\ \sigma''_2 \equiv \sigma_2 \qquad (Q)\ \sigma'_1 \dot{\sqsubseteq}^{\emptyset} \sigma''_3 \qquad (Q)\ \sigma''_3 \equiv \sigma_3$$

By Property i, there exists σ_4 such that $(Q)\ \sigma''_2 \sqsubseteq \sigma_4$ and $(Q)\ \sigma''_3 \sqsubseteq \sigma_4$. Consequently, $(Q)\ \sigma_2 \sqsubseteq \sigma_4$ holds by A-Equiv and R-Trans. Similarly, $(Q)\ \sigma_3 \sqsubseteq \sigma_4$ holds. This is the expected result.

Property iii: Intuitively, there are no critical pair in this configuration. By hypothesis, we have $(Q)\ \sigma_1 \dot{\sqsubset} \sigma_2$ **(1)** and $(Q)\ \sigma_1 \dot{\sqsubseteq}^{\bar{\alpha}} \sigma_3$ **(2)**. All rules available to derive (1) are of the form $(Q)\ C_f(\sigma'_1) \dot{\sqsubset} C_f(\sigma'_2)$, thus there exist a flexible context C_f and types σ'_1 and σ'_2 such that we have $\sigma_1 = C_f(\sigma'_1)$ **(3)** and $\sigma_2 = C_f(\sigma'_2)$. Similarly, there exist a rigid context C_r and types σ''_1 and σ'_3 such that we have $\sigma_1 = C_r(\sigma''_1)$ **(4)** and $\sigma_3 = C_r(\sigma'_3)$. We proceed by case analysis on the pair (C_f, C_r). Since σ_1 is in normal form, C_f cannot be of the form $\forall\,(Q_1, \alpha = C'_f)\ \sigma_\alpha$ and C_r cannot be of the form $\forall\,(Q_1, \alpha \geq C'_r)\ \sigma_\alpha$, with $\mathsf{nf}(\sigma_\alpha) = \alpha$. By (3) and (4), we must always have

$\sigma_1 = C_f(\sigma'_1) = C_r(\sigma''_1)$ (**5**). Additionally, Rule StSh-Hyp cannot be used to derive (2) because the prefix is unconstrained.

∘ Case $\forall\,(Q_1)\ [\ ], \forall\,(Q_1)\ [\ ]$: The rule used to derive (2) is either StSh-Up, or StSh-Alias. In both cases σ_1 is of the form $C_r(\forall\,(\alpha = \sigma_a)\ \sigma_b)$ (**6**). Hence, σ_1 must be of the form $\forall\,(Q_1)\ (\forall\,(\alpha = \sigma_a)\ \sigma_b)$. If Rule S-Hyp is used to derive (1), then σ_1 must be of the form $C_f(\sigma')$, where $(Q)\ \bot \equiv \widehat{C_f}(\sigma')$ (because Q is unconstrained). By Property 1.5.11.vi (page 54), we must have $\sigma'/\epsilon = \bot$. Hence, $\sigma' \equiv \bot$ by Property 2.1.5.iii (page 67). Moreover, σ' is in normal form since σ_1 is in normal form. Hence, σ' must be $\bot$. In summary, σ_1 is $\forall\,(Q_1)\ \bot$, which is a contradiction with (6). Hence, Rule S-Hyp cannot be used. All other rules available to derive (1) imply that σ_1 is of the form $C_f(\forall\,(\alpha \geq \sigma_a)\ \sigma_b)$. Consequently, we should have $\forall\,(Q_1)\ \forall\,(\alpha \geq \sigma_a)\ \sigma_b = \forall\,(Q_1)\ \forall\,(\alpha = \sigma_a)\ \sigma_b$, which is a contradiction. Thus this case cannot occur.

∘ Case $(\forall\,(Q_1, \alpha \geq C'_f)\ \sigma_0, \forall\,(Q_1, \alpha \geq \sigma_a)\ C'_r)$: By (5), we must have $\sigma_a = C'_f(\sigma'_1)$ and $\sigma_0 = C'_r(\sigma''_1)$. Hence, σ_1 is $\forall\,(Q_1, \alpha \geq C'_f(\sigma'_1))\ C'_r(\sigma''_1)$. We get the expected result by taking $\sigma_4 = \forall\,(Q_1, \alpha \geq C'_f(\sigma'_2))\ C'_r(\sigma'_3)$.

∘ Case $(\forall\,(Q_1, \alpha = \sigma_a)\ C'_f, \forall\,(Q_1, \alpha = C'_r)\ \sigma_0)$: By (5), we must have $\sigma_a = C'_r(\sigma''_1)$ and $\sigma_0 = C'_f(\sigma'_1)$. Hence, σ_1 is $\forall\,(Q_1, \alpha = C'_r(\sigma''_1))\ C'_f(\sigma'_1)$. We get the expected result by taking $\sigma_4 = \forall\,(Q_1, \alpha = C'_r(\sigma'_3))\ C'_f(\sigma'_2)$.

∘ Case $\forall\,(Q_1)\ [\ ], \forall\,(Q_1)\ C'_r$ (the case where C'_r is $[\ ]$ has already been discarded): By (5), we must have $\sigma'_1 = C'_r(\sigma''_1)$ (**7**). We consider the rule used to derive (1):

Subcase S-Hyp: Necessarily, σ_1 is of the form $\forall\,(Q_1)\ \bot$. This is a contradiction with $\sigma_1 = \forall\,(Q_1)\ C'_r(\sigma''_1)$ and C'_r different from $[\ ]$.

Subcase S-Up: we have $\sigma'_1 = \forall\,(\beta \geq \forall\,(Q', \alpha \diamond \sigma)\ \sigma')\ \sigma''$. By (7), we must have $C'_r = \forall\,(\beta \geq \forall\,(Q', \alpha \diamond \sigma)\ \sigma')\ C''_r$ and $C''_r(\sigma''_1) = \sigma''$. We get the expected result by taking $\sigma_4 = \forall\,(Q_1)\ \forall\,(\alpha \diamond \sigma)\ \forall\,(\beta \geq \forall\,(Q')\ \sigma')\ C''_r(\sigma'_3)$.

Subcase S-Alias: We have $\sigma'_1 = \forall\,(\alpha_1 \geq \sigma_a, Q', \alpha_2 \geq \sigma_b)\ \sigma'$. If C'_r is of the form $\forall\,(Q_1, \alpha = C''_r)\ \sigma_0$ with $\alpha \in \mathsf{dom}(Q')$ or $\alpha_2 \in \mathsf{dom}(Q_1)$, we get the expected result by taking $\sigma_4 = \forall\,(Q'_1, \alpha = C''_r(\sigma'_3))\ \sigma'_0$, and Q'_1 and σ'_0 are, respectively, Q_1 and σ_0, where the binding $(\alpha_2 \geq \sigma_b)$ is replaced by $(\alpha_2 = \alpha_1)$. Otherwise, C'_r is of the form $\forall\,(Q_2)\ [\ ]$. We proceed by case on the rule used to derive (2).

- Rule StSh-Up: we have $\sigma''_1 = \forall\,(\beta = \forall\,(Q_a, \alpha \diamond \sigma)\ \sigma')\ \sigma''$. Hence, by (7), we get $\forall\,(\alpha_1 \geq \sigma_a, Q', \alpha_2 \geq \sigma_b)\ \sigma' = \forall\,(Q_2, \beta = \forall\,(Q_a, \alpha \diamond \sigma)\ \sigma')\ \sigma''$. Necessarily, β is in $\mathsf{dom}(Q')$ or α_2 is in $\mathsf{dom}(Q_2)$. In both cases, we get the expected result by taking $\sigma_4 = \forall\,(Q'_2, \alpha \diamond \sigma, \beta = \forall\,(Q_a)\ \sigma')\ \sigma''_2$ and Q'_2 and σ''_2 are, respectively, Q_2 and σ'', where the binding $(\alpha_2 \geq \sigma_b)$ is replaced by $(\alpha_2 = \alpha_1)$.

- Rule StSh-Alias: We have $\sigma''_1 = \forall\,(\alpha'_1 = \sigma'_a, Q'', \alpha'_2 = \sigma'_b)\ \sigma''$. By (7), we get $\forall\,(\alpha_1 \geq \sigma_a, Q', \alpha_2 \geq \sigma_b)\ \sigma' = \forall\,(Q_2, \alpha'_1 = \sigma'_a, Q'', \alpha'_2 = \sigma'_b)\ \sigma''$. Necessarily, α_2 is in $\mathsf{dom}(Q_2) \cup \mathsf{dom}(Q'')$ or α'_1 and α'_2 are in $\mathsf{dom}(Q')$. In all cases, we get the expected result by taking $\forall\,(Q'_2, \alpha'_1 = \sigma'_a, Q''_2, \alpha'_2 = \alpha'_1)\ \sigma''_2$ and Q'_2, Q''_2 and σ''_2 are, respectively, Q_2, Q'' and σ'', where the binding $(\alpha_2 \geq \sigma_b)$ is replaced by $(\alpha_2 = \alpha_1)$.

SUBCASE S-NIL: We have $\sigma'_1 = \bot$. By (7), we must have $\bot = C'_r(\sigma''_1)$, which is a contradiction, since C'_r is not $[\,]$.

SUBCASE S-RIGID: We have $\sigma'_1 = \forall\,(\alpha \geq \sigma')\ \sigma$. By (7), we must have $C'_r = \forall\,(\alpha \geq \sigma')\ C''_r$ and $C''_r(\sigma''_1) = \sigma$. We get the expected result by taking $\sigma_4 = \forall\,(\alpha = \sigma')\ C''_r(\sigma'_3)$.

◦ CASE $\forall\,(Q_1)\ C'_f, \forall\,(Q_1)\ [\,]$ (the case where C'_f is $[\,]$ has already been discarded): By (5), we must have $\sigma''_1 = C'_f(\sigma'_1)$. We consider the rule used to derive (2):

SUBCASE STSH-UP: similar to S-UP above.

SUBCASE STSH-ALIAS: similar to S-ALIAS above.

Property iv: Intuitively, there are no critical pair in this configuration. By hypothesis, we have $(Q)\ C_f(\sigma_1)\ \dot{\sqsubseteq}^{\bar{\alpha}}\ \sigma_3\ (\underline{\mathbf{1}})$ and $(Q\overline{C_f})\ \sigma_1\ \dot{\sqsubseteq}^{\bar{\alpha}}\ \sigma_2$. From (1), we have $C_f(\sigma_1) = C_r(\sigma'_1)\ (\underline{\mathbf{2}})$ for some C_r and σ'_1, and σ_3 is $C_r(\sigma'_3)$. We proceed by case analysis on the pair (C_f, C_r):

◦ CASE C_f and C_r are disjoint: By Lemma 2.4.6 (page 76) and (2), there exists a two-hole context C^2 such that $C_f = C^2([\,], \sigma'_1)$ and $C_r = C^2(\sigma_1, [\,])$. Taking $\sigma_4 = C^2(\sigma_2, \sigma'_3)$ gives the expected result.

◦ CASE C_r is of the form $C_f(C')$ is not possible since $\mathsf{level}(C_f) > 1$.

◦ CASE C_f is of the form $C_r(C'_f)$: necessarily, $\mathsf{level}(C_r) = 1$, that is, by Property 2.4.4.i (page 74), C_r is of the form $\forall\,(Q_1)\ [\,]$. Besides, $\mathsf{level}(C'_f) > 1$. The hypothesis (1) can be written this way: $(Q)\ \forall\,(Q_1)\ C'_f(\sigma_1)\ \dot{\sqsubseteq}^{\bar{\alpha}}\ \forall\,(Q_1)\ \sigma'_3\ (\underline{\mathbf{3}})$. Two rules can be used to derive (3):

SUBCASE STSH-UP: Then $C'_f(\sigma_1) = \forall\,(\beta = \forall\,(Q', \alpha = \sigma)\ \sigma')\ \sigma''$. Necessarily, C'_f is of the form $\forall\,(\beta = \forall\,(Q', \alpha = \sigma)\ \sigma')\ C''_f$. Then we get the expected result by taking $\sigma_4 = \forall\,(Q_1, \alpha = \sigma, \beta = \forall\,(Q')\ \sigma')\ C''_f(\sigma_2)$

SUBCASE STSH-ALIAS: Then $C'_f(\sigma_1) = \forall\,(\alpha_1 = \sigma, Q', \alpha_2 = \sigma')\ \sigma''$ with $\sigma \equiv \sigma'$, and σ_3 is $\forall\,(\alpha_1 = \sigma, Q', \alpha_2 = \alpha_1)\ \sigma''$. Necessarily, C'_f is of the form $\forall\,(\alpha_1 = \sigma)\ \forall\,(Q_0)\ \forall\,(\alpha \geq C''_f)\ \sigma_0$, where $\alpha \in \mathsf{dom}(Q')$ or $\alpha_2 \in \mathsf{dom}(Q_0)$. In both cases, we get the expected result by taking $\sigma_4 = \forall\,(Q)\ \forall\,(\alpha_1 = \sigma, Q'_0, \alpha \geq C''_f(\sigma_2))\ \sigma'_0$ and Q'_0 and σ'_0 are, respectively, Q_0 and σ_0 where the binding $(\alpha_2 = \sigma')$ is replaced by $(\alpha_2 = \alpha_1)$.

■

Proof of Property 3.2.2

Property i: Directly, if we have $Q_1 \equiv^I Q_2$, then, for any σ such that $\mathsf{ftv}(\sigma) \subseteq I$, we have $\forall\,(Q_1)\ \sigma \equiv \forall\,(Q_2)\ \sigma$, which implies $\forall\,(Q_1)\ \sigma \sqsubseteq \forall\,(Q_2)\ \sigma\ (\underline{\mathbf{1}})$ by I-EQUIV*, as well as $\forall\,(Q_2)\ \sigma \sqsubseteq \forall\,(Q_1)\ \sigma\ (\underline{\mathbf{2}})$. By definition, (1) implies $Q_1 \sqsubseteq^I Q_2$, and (2) implies

$Q_2 \sqsubseteq^I Q_1$. Conversely, let σ be such that $\mathsf{ftv}(\sigma) \subseteq I$. By hypothesis, $Q_1 \sqsubseteq^I Q_2$ and $Q_2 \sqsubseteq^I Q_1$ hold, thus we have by definition $\forall\,(Q_1)\ \sigma \sqsubseteq \forall\,(Q_2)\ \sigma$ and $\forall\,(Q_2)\ \sigma \sqsubseteq \forall\,(Q_1)\ \sigma$. By Property 2.7.7.i (page 96), we have $\forall\,(Q_1)\ \sigma \equiv \forall\,(Q_2)\ \sigma$. Hence, by definition, $Q_1 \equiv^I Q_2$ holds.

Property ii: By hypothesis, we have a derivation of $\forall\,(Q_1)\ \bot \approx \forall\,(Q_2)\ \bot$ $(\underline{\mathbf{3}})$. We show by induction on the derivation of (3) that there exists a derivation of $\forall\,(Q_1)\ \sigma \approx \forall\,(Q_2)\ \sigma$ $(\underline{\mathbf{4}})$ for any σ. Actually, we get a derivation of (4) by replacing $\bot$ by σ in the derivation of (3). Then by Property 1.5.3.iii (page 49), we have $\forall\,(Q_1)\ \sigma \equiv \forall\,(Q_2)\ \sigma$. Hence, by definition, $Q_1 \equiv Q_2$ holds. ■

Proof of Property 3.2.3

Property i: If $(Q)\ \alpha \equiv Q(\alpha)$ holds, then we conclude by I-Equiv* and R-Trans. From now on, we assume $(Q)\ \alpha \equiv Q(\alpha)$ does not hold. This implies $\widehat{Q}(\alpha) \in \mathcal{V}$ $(\underline{\mathbf{1}})$ by Property 2.2.2.iii (page 69). By hypothesis, $(Q)\ \sigma \mathrel{-\!\!\!\!\sqsubseteq} \alpha$ holds. By Property 2.6.2.i (page 85), we have a derivation of $(Q)\ \sigma\ (\equiv\dot{\mathrel{-\!\!\!\!\sqsubseteq}}^{\emptyset})^*\ \alpha$. Assume we proved the following property:

> *If we have* $(Q)\ \sigma_1\ (\equiv\dot{\mathrel{-\!\!\!\!\sqsubseteq}}^{\emptyset})\ \sigma_2$ *and* $(Q)\ \sigma_2 \equiv \alpha$*, then we have either* $(Q)\ \sigma_1 \equiv \alpha$*, or* $(Q)\ \sigma_1 \mathrel{-\!\!\!\!\sqsubseteq} Q(\alpha)$*.*

Then we may conclude by induction on the size of $(Q)\ \sigma\ (\equiv\dot{\mathrel{-\!\!\!\!\sqsubseteq}}^{\emptyset})^*\ \alpha$, Property 2.6.2.i (page 85) and R-Trans. Therefore, it only remains to prove the property above. We reason by case analysis. By hypothesis, we have $(Q)\ \sigma_2 \equiv \alpha$ $(\underline{\mathbf{2}})$ and $(Q)\ \sigma_1\ (\equiv\dot{\mathrel{-\!\!\!\!\sqsubseteq}}^{\emptyset})\ \sigma_2$. If $(Q)\ \sigma_1 \equiv \sigma_2$ $(\underline{\mathbf{3}})$, then $(Q)\ \sigma_1 \equiv \alpha$ holds by R-Trans, (3) and (2), and the result is shown. Otherwise, by definition of $(\equiv\dot{\mathrel{-\!\!\!\!\sqsubseteq}}^{\emptyset})$, there exist σ_1' and σ_2' such that $(Q)\ \sigma_1 \equiv \sigma_1'$ $(\underline{\mathbf{4}})$, $(Q)\ \sigma_1'\ \dot{\mathrel{-\!\!\!\!\sqsubseteq}}^{\bar{\alpha}}\ \sigma_2'$ $(\underline{\mathbf{5}})$ and $(Q)\ \sigma_2' \equiv \sigma_2$ $(\underline{\mathbf{6}})$ hold. We have $(Q)\ \sigma_2' \equiv \alpha$ by R-Trans, (6), and (2). Hence, we have $\widehat{Q}(\sigma_2') \equiv \widehat{Q}(\alpha)$ by Corollary 1.5.10. By (1), this means $\widehat{Q}(\sigma_2') \equiv \beta$, where $\beta = \widehat{Q}(\alpha)$ $(\underline{\mathbf{7}})$. Hence, $\mathsf{nf}(\widehat{Q}(\sigma_2'))$ is β by Property 1.5.11.i (page 54). We get $\widehat{Q}(\mathsf{nf}(\sigma_2')) = \beta$ by Property 1.5.6.iii (page 51), which implies $\mathsf{nf}(\sigma_2') = \gamma$ $(\underline{\mathbf{8}})$ for some variable γ such that $\widehat{Q}(\gamma) = \beta$ $(\underline{\mathbf{9}})$. Three rules are available to derive (5): However, rules StSh-Up and StSh-Alias cannot be used to derive (5), because the side-conditions of these rules prevent σ_2' from being in $\mathcal{V}$. Thus, Rule StSh-Hyp must be used. Then σ_1' is $C_r(\sigma')$ $(\underline{\mathbf{10}})$, σ_2' is $C_r(\alpha')$ $(\underline{\mathbf{11}})$, with $(\alpha' = \sigma_0) \in Q$ $(\underline{\mathbf{12}})$ and $(Q)\ \sigma_0 \equiv \widehat{C_r}(\sigma')$ $(\underline{\mathbf{13}})$. Additionally, $\alpha' \notin \mathsf{dom}(C_r)$ $(\underline{\mathbf{14}})$ and C_r is necessarily useful (otherwise, we would have $(Q)\ \sigma_1' \equiv \sigma_2'$). We have $(Q)\ C_r(\sigma') \equiv C_r(\widehat{C_r}(\sigma'))$ $(\underline{\mathbf{15}})$ by Eq-Mono, R-Context-L, and R-Context-R. From (13), we get $(Q)\ C_r(\widehat{C_r}(\sigma')) \equiv C_r(\sigma_0)$ $(\underline{\mathbf{16}})$. By R-Trans, (10), (15), and (16), we get $(Q)\ \sigma_1' \equiv C_r(\sigma_0)$ $(\underline{\mathbf{17}})$. By (14), we get $\alpha' \in \mathsf{ftv}(C_r(\alpha'))$, that is $\alpha' \in \mathsf{ftv}(\sigma_2')$, which implies $\alpha' \in \mathsf{ftv}(\mathsf{nf}(\sigma_2'))$

by Property 1.5.6.ii (page 51). By (8), this means that α' is γ. Consequently, $\sigma_0 = Q(\gamma)$ $\mathbf{(\underline{18})}$ holds from (12). By Property 2.4.4.v (page 74), (8), and (11), we must have $\mathsf{level}(C_r) \leqslant 1$, that is $\mathsf{level}(C_r) = 1$ $\mathbf{(\underline{19})}$ since C_r is useful (Property 2.4.3.iii (page 73)). We have $\mathsf{dom}(C_r) \# \mathsf{ftv}(\sigma_0)$ $\mathbf{(\underline{20})}$ by well-formedness of (12). By (19) and (20), we get $(Q)\ C_r(\sigma_0) \equiv \sigma_0$ $\mathbf{(\underline{21})}$ by Property 2.4.4.ii (page 74) and Eq-Free. Finally, this gives $(Q)\ \sigma_1 \equiv Q(\gamma)$ by (4), (17), (21), (18), and R-Trans. We get $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!-} Q(\gamma)$ $\mathbf{(\underline{22})}$ by A-Equiv. By Property 2.2.2.ii (page 69) applied to (7) and (9), we have $Q(\alpha) = Q(\beta)$ and $Q(\beta) = Q(\gamma)$. Therefore, (22) gives $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!-} Q(\alpha)$, which is the expected result.

Property ii: If $(Q)\ \sigma \mathrel{\sqsubseteq\!\!\!\!-} \alpha$ holds, then Property i and I-Abstract give the expected result. Hence, we assume $(Q)\ \sigma \mathrel{\sqsubseteq\!\!\!\!-} \alpha$ does not hold. This implies $\sigma \notin \mathcal{T}$ by Lemma 2.1.6. Furthermore, we also assume that $(Q)\ \alpha \equiv Q(\alpha)$ does not hold (if it holds, we get the result by I-Equiv* and R-Trans). This implies $\widehat{Q}(\alpha) \in \mathcal{V}$ $\mathbf{(\underline{1})}$ by Property 2.2.2.iii (page 69). By Property 2.6.2.ii (page 85), we have a derivation of $(Q)\ \sigma\ (\equiv\dot\sqsubseteq)^*\ \alpha$. Assume we proved the following property:

> *If we have* $(Q)\ \sigma_1\ (\equiv\dot\sqsubseteq)\ \sigma_2$ *and* $(Q)\ \sigma_2 \equiv \alpha$*, then we have either* $(Q)\ \sigma_1 \equiv \alpha$*, or* $(Q)\ \sigma_1 \sqsubseteq Q(\alpha)$*.*

Then we conclude by induction on the size of $(Q)\ \sigma\ (\equiv\dot\sqsubseteq)^*\ \alpha$, Property 2.6.2.ii (page 85) and R-Trans. Therefore, it remains only to show the property above. We reason by case analysis. By hypothesis, we have $(Q)\ \sigma_2 \equiv \alpha$ $\mathbf{(\underline{2})}$ and $(Q)\ \sigma_1\ (\equiv\dot\sqsubseteq)\ \sigma_2$. If $(Q)\ \sigma_1 \equiv \sigma_2$ $\mathbf{(\underline{3})}$, then $(Q)\ \sigma_1 \equiv \alpha$ holds by R-Trans, (3) and (2), and the result is shown. Otherwise $\mathbf{(\underline{4})}$ there exist σ_1' and σ_2' such that $(Q)\ \sigma_1 \equiv \sigma_1'$ $\mathbf{(\underline{5})}$, $(Q)\ \sigma_1' \dot\sqsubseteq \sigma_2'$ $\mathbf{(\underline{6})}$ and $(Q)\ \sigma_2' \equiv \sigma_2$ $\mathbf{(\underline{7})}$. We have $(Q)\ \sigma_2' \equiv \alpha$ $\mathbf{(\underline{8})}$ by R-Trans, (7), and (2). Hence, we have $\widehat{Q}(\sigma_2') \equiv \widehat{Q}(\alpha)$ by Corollary 1.5.10. By (1), this means $\widehat{Q}(\sigma_2') \equiv \beta$, where $\beta = \widehat{Q}(\alpha)$ $\mathbf{(\underline{9})}$. Hence, $\mathsf{nf}(\widehat{Q}(\sigma_2'))$ is β by Property 1.5.11.i (page 54). We get $\widehat{Q}(\mathsf{nf}(\sigma_2')) = \beta$ by Property 1.5.6.iii (page 51), which implies $\mathsf{nf}(\sigma_2') = \gamma$ $\mathbf{(\underline{10})}$ for some variable γ such that $\widehat{Q}(\gamma) = \beta$ $\mathbf{(\underline{11})}$ holds. Three rules are available to derive (6):

○ Case C-Strict: Then we have $(Q)\ \sigma_1' \dot\sqsubset \sigma_2'$ $\mathbf{(\underline{12})}$ and $\sigma_2' \in \mathcal{V}$ $\mathbf{(\underline{13})}$ (by (10)). Rules S-Up, S-Alias, S-Rigid cannot be used to derive (12), because the side-conditions of these rules prevent σ_2' from being in $\mathcal{V}$. If Rule S-Nil is used, then there exists a useful context C_f and a closed type σ such that σ_2' is $C_f(\sigma)$. Since $\sigma_2' \in \mathcal{V}$, this implies $\sigma \in \mathcal{V}$, which is a contradiction with $\mathsf{ftv}(\sigma) = \emptyset$. Hence this rule cannot be used. Last, Rule S-Hyp is used to derive (12), and σ_1' is $C_f(\sigma')$ $\mathbf{(\underline{14})}$, σ_2' is $C_f(\alpha')$ $\mathbf{(\underline{15})}$, with $(\alpha' \geq \sigma_0) \in Q$ $\mathbf{(\underline{16})}$ and $(Q)\ \sigma_0 \equiv \widehat{C_f}(\sigma')$ $\mathbf{(\underline{17})}$. Additionally, $\alpha' \notin \mathsf{dom}(C_f)$ $\mathbf{(\underline{18})}$, and C_f is necessarily useful (otherwise, we would have $(Q)\ \sigma_1' \equiv \sigma_2'$). We have $(Q)\ C_f(\sigma') \equiv C_f(\widehat{C_f}(\sigma'))$ $\mathbf{(\underline{19})}$ by Eq-Mono, R-Context-L, and R-Context-R. By (17), we have $(Q)\ C_f(\widehat{C_f}(\sigma')) \equiv C_f(\sigma_0)$ $\mathbf{(\underline{20})}$. By R-Trans, (14), (19), and (20), we get $(Q)\ \sigma_1' \equiv C_f(\sigma_0)$ $\mathbf{(\underline{21})}$. By (18), we get $\alpha' \in \mathsf{ftv}(C_f(\alpha'))$, that is $\alpha' \in \mathsf{ftv}(\sigma_2')$, which implies $\alpha' \in \mathsf{ftv}(\mathsf{nf}(\sigma'_2))$ by Property 1.5.6.ii (page 51). By (10), this means that α' is γ. Consequently, $\sigma_0 = Q(\gamma)$ **(<u>22</u>)**. By Property 2.4.4.v (page 74), (13), and (15), we must have $\mathsf{level}(C_f) \leqslant 1$, that is $\mathsf{level}(C_f) = 1$ **(<u>23</u>)** since C_f is useful (Property 2.4.3.iii (page 73)). We have $\mathsf{dom}(C_f) \# \mathsf{ftv}(\sigma_0)$ **(<u>24</u>)** by well-formedness of (16). By (23) and (24), we get $(Q)\ C_f(\sigma_0) \equiv \sigma_0$ **(<u>25</u>)** by Property 2.4.4.ii (page 74) and Eq-Free. Finally, this gives $(Q)\ \sigma_1 \equiv Q(\gamma)$ by (5), (21), (25), (22), and R-Trans. We get $(Q)\ \sigma_1 \sqsubseteq Q(\gamma)$ **(<u>26</u>)** by I-Equiv*. By Property 2.2.2.ii (page 69) applied to (9) and (11), we have $Q(\alpha) = Q(\beta)$ and $Q(\beta) = Q(\gamma)$. Therefore, (26) gives $(Q)\ \sigma_1 \sqsubseteq Q(\alpha)$, which is the expected result.

∘ Case C-Abstract-F: We must have σ'_2 of the form $C_f(\sigma_0)$ with $\mathsf{level}(C_f) > 1$. By (10), $\sigma'_2 \in \mathcal{V}$. By Property 2.4.4.v (page 74), this implies $\mathsf{level}(C_f) \leqslant 1$, which is a contradiction. Therefore, this case cannot occur.

∘ Case C-Abstract-R: Then by hypothesis, $(Q)\ \sigma'_1 \mathrel{\sqsubseteq\!\!\!\!-} \sigma'_2$ **(<u>27</u>)** holds, thus (Q) $\sigma_1 \mathrel{\sqsubseteq\!\!\!\!-} \alpha$ holds by R-Trans, A-Equiv and (5), (27) and (8). We assumed (in (4)), that $(Q)\ \sigma_1 \equiv \sigma_2$ does not hold. This implies $\sigma_1 \notin \mathcal{T}$ by Lemma 2.1.6. Hence, $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!-} Q(\alpha)$ by Property i, which gives $(Q)\ \sigma_1 \sqsubseteq Q(\alpha)$ by I-Abstract. This is the expected result.

<u>Property iii</u>: We have $(Q)\ Q(\alpha) \sqsubseteq Q[\alpha]$ by I-Hyp or A-Hyp and I-Abstract. We conclude by Property 2.2.2.i (page 69). ■

Proof of Property 3.3.2

<u>Property i</u> : It is shown by observing that $\alpha \in \mathsf{ftv}(\forall\,(Q_2)\ \nabla_{I \cup J})$ is equivalent to $\alpha \in \mathsf{ftv}(\forall\,(Q_2)\ \nabla_I) \cup \mathsf{ftv}(\forall\,(Q_2)\ \nabla_J)$.
<u>Property ii</u> : It suffices to show the result for a single commutation. Then the result follows by immediate induction on the number of commutations. Hence, it suffices to show that $\mathsf{dom}(Q_1, \alpha \diamond \sigma, \alpha' \diamond' \sigma', Q_0/\bar{\alpha})$ and $\mathsf{dom}(Q_1, \alpha' \diamond' \sigma', \alpha \diamond \sigma, Q_0/\bar{\alpha})$ are equal when $\alpha \notin \mathsf{ftv}(\sigma')$ and $\alpha' \notin \mathsf{ftv}(\sigma)$. We get the expected result by observing that α is in $\mathsf{ftv}(\forall\,(\alpha' \diamond' \sigma', Q_0)\ \nabla_I)$ if and only if α is in $\mathsf{ftv}(\forall\,(Q_0)\ \nabla_I)$, and similarly, α' is in $\mathsf{ftv}(\forall\,(\alpha \diamond \sigma, Q_0)\ \nabla_I)$ if and only if α' is in $\mathsf{ftv}(\forall\,(Q_0)\ \nabla_I)$. ■

Proof of Property 3.4.2

<u>Property i</u> : It is shown by induction on the derivation of $Q_1 \Diamond_\ell^I Q_2$. All cases are easy.

<u>Property ii</u>: We say that α appears in a judgment $Q_1 \Diamond_\ell^I Q_2$ if α is in $\mathsf{dom}(Q_1) \cup \mathsf{dom}(Q_2)$. We say that α appears in a derivation when it appears in one of its judgments. By hypothesis, we have a derivation of $Q \Diamond_\ell^I Q'$ **(<u>1</u>)**. We show that $\phi(Q) \Diamond_\ell^{\phi(I)} \phi(Q')$ holds for a renaming ϕ such that $\mathsf{codom}(\phi)$ is fresh, that is, disjoint from the set of type variables appearing in the derivation of (1). All cases are easy. Then any renaming

ϕ can be decomposed into $\phi_1 \circ \phi_2$ such that ϕ_2 is fresh, relatively to the derivation of $Q \sqsubseteq_\ell^I Q'$, and ϕ_1 is fresh, relatively to the derivation of $\phi_2(Q) \sqsubseteq_\ell^{\phi_2(I)} \phi_2(Q')$.

Property iii: By hypothesis, we have a derivation of $Q_1 \Diamond_\ell^I Q_2$ $(\underline{\mathbf{1}})$. Let ϕ be a renaming mapping the domain $\mathsf{dom}(Q) - I$ to fresh variables (that is, outside $\mathsf{dom}(QQ_1Q_2)$). By Property ii, we have a derivation of $\phi(Q_1) \Diamond_\ell^{\phi(I)} \phi(Q_2)$ $(\underline{\mathbf{2}})$. Since $\mathsf{dom}(\phi) \# I$, we have $\phi(Q_1) \Diamond_\ell^I \phi(Q_2)$ $(\underline{\mathbf{3}})$ from (2). Besides, by hypothesis, we have $\mathsf{dom}(Q) \# \mathsf{dom}(Q_1) \cup \mathsf{dom}(Q_2)$ $(\underline{\mathbf{4}})$, hence ϕ is invariant on Q_1 and Q_2. Then (3) is written $Q_1 \Diamond_\ell^I Q_2$ $(\underline{\mathbf{5}})$. The difference between (1) and (5) lies in the fact that intermediate variables (variables introduced by PE-Free) are renamed by ϕ in (5), and therefore such variables are outside $\mathsf{dom}(Q)$. By hypothesis $\mathsf{utv}(Q) \subseteq I$, thus $\mathsf{utv}(Q) \subseteq \mathsf{dom}(Q_1) \cap \mathsf{dom}(Q_2)$ $(\underline{\mathbf{6}})$ by well-formedness of (5). Hence, by (4) and (6), Q_1Q and Q_2Q are well-formed and closed. The proof is by induction on the derivation of (5). Case PE-Refl is immediate. We show directly cases PE-Mono, PE-Swap, PE-Comm, PE-Context-L, PA-Context-L, PI-Context-L and PI-Rigid by replacing Q_0 by Q_0Q. Cases PA-Equiv and PI-Abstract are by induction hypothesis.

∘ Case PE-Free: We have $Q_2 = (Q_1, \alpha \diamond \sigma)$, with $\alpha \notin \mathsf{dom}(Q_1)$ and $\alpha \notin I$ $(\underline{\mathbf{7}})$. Thanks to the renaming ϕ, we know that $\alpha \notin \mathsf{dom}(Q)$. Besides, (7) and the hypothesis $\mathsf{utv}(Q) \subseteq I$ imply that $\alpha \notin \mathsf{utv}(Q)$. Hence $Q_1Q \equiv_\ell (Q_1, \alpha \diamond \sigma, Q)$ holds by PE-Free and PE-Comm. This is the expected result.

∘ Case PE-Trans, PA-Trans and PI-Trans: We have $Q_1 \Diamond_\ell^I Q_1'$ and $Q_1' \Diamond_\ell^I Q_2$. By induction hypothesis, we get $Q_1Q \Diamond_\ell^{I \cup \mathsf{dom}(Q)} Q_1'Q$ and $Q_1'Q \Diamond_\ell^{I \cup \mathsf{dom}(Q)} Q_2Q$. By PE-Trans, PA-Trans, or PI-Trans, we get $Q_1Q \Diamond_\ell^{I \cup \mathsf{dom}(Q)} Q_2Q$.

Property iv: By hypothesis, Q is closed $(\underline{\mathbf{1}})$ and well-formed, and ϕ is a renaming of $\mathsf{dom}(Q)$. This implies that ϕ is injective on $\mathsf{dom}(Q)$. Hence, $\phi(Q)$ is closed and well-formed. Additionally, $\mathsf{dom}(\phi) \subseteq \mathsf{dom}(Q)$ holds by hypothesis, thus $\mathsf{codom}(\phi) \subseteq \mathsf{dom}(\phi(Q))$. As a consequence, $\phi(Q)\underline{\phi}$ is closed and well-formed. Let I be $\mathsf{dom}(Q)$ $(\underline{\mathbf{2}})$. The proof is by induction on the size of the finite set $\mathsf{dom}(\phi)$. If this set is empty, then $Q = \phi(Q)$ and we get the result by PE-Refl. Otherwise, let $(\alpha \diamond \sigma)$ be the leftmost binding of Q such that $\alpha \in \mathsf{dom}(\phi)$. We can write Q in the form $(Q_1, \alpha \diamond \sigma, Q_2)$, and $\mathsf{dom}(Q_1) \# \mathsf{dom}(\phi)$ $(\underline{\mathbf{3}})$. Since Q is closed by (1), we have $\mathsf{ftv}(\sigma) \subseteq \mathsf{dom}(Q_1)$, hence $\mathsf{ftv}(\sigma) \# \mathsf{dom}(\phi)$ by (3). As a consequence, we have $\phi(\sigma) = \sigma$ $(\underline{\mathbf{4}})$. We write α' for $\phi(\alpha)$ $(\underline{\mathbf{5}})$. Since ϕ is a renaming of $\mathsf{dom}(Q)$, we have $\mathsf{dom}(Q) \# \mathsf{codom}(\phi)$, which implies $\alpha' \notin \mathsf{dom}(Q)$ $(\underline{\mathbf{6}})$. We have the following:

$$\begin{array}{lll}
Q = & (Q_1, \alpha \diamond \sigma, Q_2) & \text{by notation} \\
\equiv^I & (Q_1, \alpha \diamond \sigma, \alpha' = \alpha, Q_2) & \text{by PE-Free and (6)} \\
\equiv^I & (Q_1, \alpha' \diamond \sigma, \alpha = \alpha', Q_2) & \text{by PE-Swap} \\
\equiv^I & (Q_1, \alpha' \diamond \sigma, \alpha = \alpha', Q_2[\alpha'/\alpha]) & \text{by PE-Mono}^\star \\
\equiv^I & (Q_1, \alpha' \diamond \sigma, Q_2[\alpha'/\alpha], \alpha = \alpha') & \text{by PE-Comm}
\end{array}$$

Let Q_1' be $(Q_1, \alpha' \diamond \sigma, Q_2[\alpha'/\alpha], \alpha = \alpha')$. We have shown that $Q \equiv^I Q_1'$ $(\underline{7})$ holds. Applying the induction hypothesis to Q_1' and on the renaming $\phi' = \phi_{|\,\mathsf{dom}(\phi)-\alpha}$, we get $Q_1' \equiv \phi'(Q_1')\underline{\phi'}$. This gives $Q_1' \equiv (Q_1, \alpha' \diamond \sigma, \phi'(Q_2[\alpha'/\alpha]), \alpha = \alpha', \underline{\phi'})$, that is, $Q_1' \equiv (Q_1, \alpha' \diamond \sigma, \phi(Q_2), \alpha = \alpha', \underline{\phi'})$. Since $(\alpha = \alpha', \underline{\phi'})$ is $\underline{\phi}$, we get $Q_1' \equiv (Q_1, \alpha' \diamond \sigma, \phi(Q_2), \underline{\phi})$. Hence, $Q_1' \equiv \phi((Q_1, \alpha \diamond \sigma, Q_2))\underline{\phi}$ holds by (4) and (5), that is, $Q_1' \equiv \phi(Q)\underline{\phi}$. By Property i, we get $Q_1' \equiv^I \phi(Q)\underline{\phi}$ $(\underline{8})$. By PE-TRANS, (7), and (8), we get $Q \equiv^I \phi(Q)\underline{\phi}$, that is, $Q \equiv \phi(Q)\underline{\phi}$ by notation and (2).

Property v: By induction on the size of $\mathsf{dom}(Q)$. If Q is empty, the result is by EQ-REFL. Otherwise, Q is $(\alpha \diamond \sigma, Q')$ and ϕ is $\phi_1 \circ \phi' = \phi' \circ \phi_1$ $(\underline{1})$, where $\mathsf{dom}(\phi') = \mathsf{dom}(Q')$, $\mathsf{dom}(\phi_1) = \{\alpha\}$ and $\phi_1(\alpha) = \alpha'$. We have

$$\begin{array}{lll}
 & Q_1 Q \phi(Q) Q_2 & \\
= & (Q_1, \alpha \diamond \sigma, Q', \alpha' \diamond \sigma, \phi(Q') Q_2) & \text{by definition} \\
\sqsubseteq & (Q_1, \alpha \diamond \sigma, Q', \alpha' = \alpha, \phi(Q') Q_2) & \text{by PI-CONTEXT-L and I-HYP or} \\
 & & \text{by PA-CONTEXT-L and A-HYP} \\
= & (Q_1, \alpha \diamond \sigma, Q' \underline{\phi_1^\neg} \phi(Q') Q_2) & \text{by definition of } \phi_1 \\
\equiv & (Q_1, \alpha \diamond \sigma, Q' \underline{\phi_1^\neg} \phi_1^\neg(\phi(Q') Q_2)) & \text{by PE-MONO}^\star \\
= & (Q_1, \alpha \diamond \sigma, Q' \underline{\phi_1^\neg} \phi'(Q') Q_2) & \text{since } \phi \text{ is } \phi_1 \circ \phi' \text{ (from (1))} \\
\equiv & (Q_1, \alpha \diamond \sigma, \underline{\phi_1^\neg} Q' \phi'(Q') Q_2) & \text{by PE-COMM} \\
\sqsubseteq & (Q_1, \alpha \diamond \sigma, \underline{\phi_1^\neg} Q' \underline{\phi'^\neg} Q_2) & \text{by induction hypothesis} \\
\equiv & (Q_1, \alpha \diamond \sigma, Q' \underline{\phi_1^\neg} \underline{\phi'^\neg} Q_2) & \text{by PE-COMM} \\
= & (Q_1, \alpha \diamond \sigma, Q' \underline{\phi^\neg} Q_2) & \text{since } \phi^\neg \text{ is } \phi_1^\neg \circ \phi'^\neg \text{ (from (1))} \\
= & Q_1 Q \underline{\phi^\neg} Q_2 &
\end{array}$$

This is the expected result.

■

Proof of Lemma 3.4.4

By Lemma 2.3.3 (page 70) and $(Q)\ \sigma_1 \diamondsuit \sigma_2$, we have a derivation of $(Q)\ \sigma_1 \diamondsuit \sigma_2$ $(\underline{1})$ which is thrifty and follows the restrictions of Lemma 2.3.1. The proof is by induction on the derivation of (1).

- ○ CASE EQ-REFL: Taking $\theta = \mathtt{id}$ gives the expected result.
- ○ CASE EQ-FREE: We have two subcases:

SUBCASE σ_1 is $\forall\,(\alpha \diamond \sigma)\ \sigma_2$: Then Q_1 is $(\alpha \diamond \sigma, Q_2)$ and τ_2 is τ_1. We take $\theta = \mathtt{id}$. Then $(QQ_2)\ \theta(\tau_1) \equiv \tau_2$ holds. Let I be $\mathsf{dom}(Q_1/\tau_1)$, that is, $\mathsf{dom}(Q_2/\tau_1)$ since $\alpha \notin \mathsf{ftv}(\forall\,(Q_2)\ \tau_1)$. This gives $I = \mathsf{dom}(Q_2/\tau_2)$, and $\theta(I) \subseteq \mathsf{dom}(Q_2/\tau_2)$. Moreover, $QQ_1 \equiv_\ell^{\mathsf{dom}(Q) \cup I} QQ_2\underline{\theta}$ holds by PE-COMM and PE-FREE since $\alpha \notin I$ and α is not free in any bound of Q_2.

Subcase σ_2 is $\forall\,(\alpha \diamond \sigma)\ \sigma_1$: similar.

∘ Case Eq-Comm: By taking $\theta = \mathtt{id}$ and by PE-Comm.

∘ Case Eq-Var: We get the expected result by taking $\theta = \mathtt{id}$, and by observing that $\mathsf{cf}(\sigma_1) = \mathsf{cf}(\sigma_2)$.

∘ Case Eq-Mono: By hypothesis, $(\alpha \diamond \sigma) \in Q$ and $(Q)\ \sigma \equiv \tau$ holds. Besides, σ_2 and σ_1 are τ_2 and τ_1 respectively. Consequently, Q_2 and Q_1 are $\emptyset$. Taking $\theta = \mathtt{id}$ gives the expected result.

∘ Case R-Context-L, R-Context-Rigid, and R-Context-Flexible: We have $\sigma_1 = \forall\,(\alpha \diamond \sigma_1')\ \sigma$ and $\sigma_2 = \forall\,(\alpha \diamond \sigma_2')\ \sigma$. The premise is $(Q)\ \sigma_1' \Diamond \sigma_2'$ $(\underline{\mathbf{2}})$. By hypothesis (Lemma 2.3.1), $\alpha \in \mathsf{ftv}(\sigma)$ and $\mathsf{nf}(\sigma)$ is not α. Hence, $\mathsf{cf}(\sigma_1)$ is $\forall\,(\alpha \diamond \sigma_1')\ \mathsf{cf}(\sigma)$ and $\mathsf{cf}(\sigma_2)$ is $\forall\,(\alpha \diamond \sigma_2')\ \mathsf{cf}(\sigma)$, thus Q_1 is $(\alpha \diamond \sigma_1', Q')$ and Q_2 is $(\alpha \diamond \sigma_2', Q')$. Then we get the expected result by taking $\theta = \mathtt{id}$ and by rules PE-Context-L, PA-Context-L or PI-Context-L, and (2).

∘ Case A-Equiv: By induction hypothesis and PA-Equiv.

∘ Case I-Hyp and A-Hyp: Then σ_2 is α, with $(\alpha \diamond \sigma) \in \mathsf{dom}(Q)$. By restrictions of Lemma 2.3.1, we must have $\sigma \notin \mathcal{T}$. Hence, $Q(\alpha) \notin \mathcal{T}$, which is a contradiction since, by hypothesis, $\mathsf{nf}(\sigma_2) \in \vartheta$ implies $Q(\mathsf{nf}(\sigma_2)) \in \mathcal{T}$. Hence, these cases cannot occur.

∘ Case I-Abstract: By induction hypothesis, and by PI-Abstract.

∘ Case R-Trans: By hypothesis, we have $(Q)\ \sigma_1 \Diamond \sigma_1'$ $(\underline{\mathbf{3}})$ and $(Q)\ \sigma_1' \Diamond \sigma_2$ $(\underline{\mathbf{4}})$. Additionally, $\mathsf{nf}(\sigma_2) \in \vartheta$ implies $Q(\mathsf{nf}(\sigma_2)) \in \mathcal{T}$ $(\underline{\mathbf{5}})$. Besides, $\mathsf{nf}(\sigma_1) \neq \bot$ $(\underline{\mathbf{6}})$ by hypothesis. By Property 2.7.7.ii (page 96) and Property 1.5.6.i (page 51) applied to (3) and (6), we get $\mathsf{nf}(\sigma_1') \neq \bot$. By Properties 2.2.2.vi (page 69) and 1.5.6.i (page 51) applied to (4) and (5), $\mathsf{nf}(\sigma_1') \in \vartheta$ implies $Q(\mathsf{nf}(\sigma_1')) \in \mathcal{T}$. Let $\forall\,(Q_1)\ \tau_1$ be $\mathsf{cf}(\sigma_1)$. By induction hypothesis on (3), there exists an alpha-conversion of $\mathsf{cf}(\sigma_1')$, written $\forall\,(Q_1')\ \tau_1'$ and a substitution θ_1 such that we have

$$(QQ_1')\ \theta_1(\tau_1) \equiv \tau_1'\ (\underline{\mathbf{7}}) \qquad I_1 \triangleq \mathsf{dom}(Q_1/\tau_1)\ (\underline{\mathbf{8}}) \qquad \mathsf{dom}(\theta_1) \subseteq I_1\ (\underline{\mathbf{9}})$$

$$\theta_1(I_1) \subseteq \mathsf{dom}(Q) \cup \mathsf{dom}(Q_1'/\tau_1')\ (\underline{\mathbf{10}}) \qquad QQ_1 \Diamond_\ell^{\mathsf{dom}(Q) \cup I_1} QQ_1'\underline{\theta_1}\ (\underline{\mathbf{11}})$$

From (9) and (8), we get $\mathsf{dom}(\theta_1) \subseteq \mathsf{dom}(Q_1)$ $(\underline{\mathbf{12}})$. By induction hypothesis and (4), there exists an alpha-conversion of $\mathsf{cf}(\sigma_2)$, written $\forall\,(Q_2)\ \tau_2$ and a substitution θ_2 such that we have

$$(QQ_2)\ \theta_2(\tau_1') \equiv \tau_2\ (\underline{\mathbf{13}}) \qquad I_2 \triangleq \mathsf{dom}(Q_1'/\tau_1')\ (\underline{\mathbf{14}}) \qquad \mathsf{dom}(\theta_2) \subseteq I_2\ (\underline{\mathbf{15}})$$

$$\theta_2(I_2) \subseteq \mathsf{dom}(Q) \cup \mathsf{dom}(Q_2/\tau_2)\ (\underline{\mathbf{16}}) \qquad QQ_1' \Diamond_\ell^{\mathsf{dom}(Q) \cup I_2} QQ_2\underline{\theta_2}\ (\underline{\mathbf{17}})$$

Let ϕ **(18)** be a renaming mapping $\mathsf{dom}(Q_2)$ to fresh variables (that is, outside the domain $\mathsf{dom}(QQ_1) \cup \mathsf{dom}(Q'_1)$). We have $\mathsf{dom}(\phi) \subseteq \mathsf{dom}(Q_2)$ **(19)** and ϕ is a renaming of $\mathsf{dom}(QQ_2)$ **(20)**. Let Q'_2 be $\phi(Q_2)$ and τ'_2 be $\phi(\tau_2)$. We note that $\forall\,(Q'_2)\ \tau'_2$ is an alpha-conversion of $\forall\,(Q_2)\ \tau_2$, that is, an alpha-conversion of $\mathsf{cf}(\sigma_2)$. Let θ'_2 be $\phi \circ \theta_2$ restricted to I_2. By definition, we have $\mathsf{dom}(\theta'_2) \subseteq I_2$ **(21)**. By Property 1.7.2.i (page 59) applied to (13), we get $(Q\phi(Q_2))\ \phi(\theta_2(\tau'_1)) \equiv \phi(\tau_2)$, that is, $(QQ'_2)\ \theta'_2(\tau'_1) \equiv \tau'_2$ **(22)**. The prefix QQ_2 is a closed well-formed prefix **(23)**. By Property 3.4.2.iv (page 106), (20) and (23), we get $QQ_2 \equiv \phi(QQ_2)\underline{\phi}$, that is, $QQ_2 \equiv Q\phi(Q_2)\underline{\phi}$ **(24)**. By well-formedness of (17), $QQ_2\underline{\theta_2}$ is well-formed. Hence, $\mathsf{dom}(\theta_2)\ \#\ \mathsf{dom}(QQ_2)$ holds (we simply write $\theta_2\ \#\ QQ_2$ **(25)**). From (19) and (25), we get $\theta_2\ \#\ \phi$ **(26)**. From (15) and (14), we get $\mathsf{dom}(\theta_2) \subseteq \mathsf{dom}(Q'_1)$. By definition of ϕ (18), this implies $\mathsf{dom}(\theta_2)\ \#\ \mathsf{codom}(\phi)$, thus $\theta_2\ \#\ \phi(Q_2)$ **(27)** holds. Hence, from (25), (26), and (27), we have $\theta_2\ \#\ Q\phi(Q_2)\underline{\phi}$ **(28)**. Additionally, $\mathsf{codom}(\theta_2) \subseteq \mathsf{dom}(Q) \cup \mathsf{dom}(Q_2)$ **(29)** holds from (16). By Property 3.4.2.iii (page 106), (28), (25), (29), and (24), we get $QQ_2\underline{\theta_2} \equiv Q\phi(Q_2)\underline{\phi}\underline{\theta_2}$ **(30)**. By PE-FREE and (30), we get $QQ_2\underline{\theta_2} \equiv^{\mathsf{dom}(Q)\cup I_2} QQ'_2\underline{\theta'_2}$ **(31)**. By PE-TRANS, (17), and (31), we have $QQ'_1 \diamondsuit_\ell^{\mathsf{dom}(Q)\cup I_2} QQ'_2\underline{\theta'_2}$ **(32)**. Applying ϕ to (16), we get $\theta'_2(I_2) \subseteq \mathsf{dom}(Q) \cup \mathsf{dom}(Q'_2/\tau'_2)$ **(33)**. By (21) and (14), we get $\mathsf{dom}(\theta'_2) \subseteq \mathsf{dom}(Q'_1)$. In summary, we have shown

$$(QQ'_2)\ \theta'_2(\tau'_1) \equiv \tau'_2\ (22) \qquad I_2 \triangleq \mathsf{dom}(Q'_1/\tau'_1)\ (14) \qquad \mathsf{dom}(\theta'_2) \subseteq I_2\ (21)$$

$$\theta'_2(I_2) \subseteq \mathsf{dom}(Q) \cup \mathsf{dom}(Q'_2/\tau'_2)\ (33) \qquad QQ'_1 \diamondsuit_\ell^{\mathsf{dom}(Q)\cup I_2} QQ'_2\underline{\theta'_2}\ (32)$$

Since (11) is well-formed, we have $\theta_1\ \#\ QQ'_1$ **(34)**. By (14), we get $I_2 \subseteq \mathsf{dom}(Q'_1)$. Hence, $\mathsf{dom}(\theta_1)\ \#\ \mathsf{dom}(Q) \cup I_2$ **(35)** holds by (34). Thus, by (21), is also gives $\mathsf{dom}(\theta_1)\ \#\ \mathsf{dom}(\theta'_2)$ **(36)**. By (12) and by definition of ϕ (18), we have $\mathsf{dom}(\theta_1)\ \#\ \mathsf{dom}(Q'_2)$ **(37)**. Hence, we have $\theta_1\ \#\ QQ'_2\theta'_2$ **(38)** from (34), (37), and (36). Additionally, $\mathsf{codom}(\theta_1) \subseteq \mathsf{dom}(Q) \cup \mathsf{dom}(Q'_1/\tau'_1)$ holds by (10). Hence, by (14), we get $\mathsf{codom}(\theta_1) \subseteq \mathsf{dom}(Q) \cup I_2$ **(39)**. In summary, we have shown

$$\theta_1\ \#\ QQ'_1\ (34) \qquad \theta_1\ \#\ QQ'_2\underline{\theta'_2}\ (38) \qquad \mathsf{codom}(\theta_1) \subseteq \mathsf{dom}(Q) \cup I_2\ (39)$$

By Property 3.4.2.iii (page 106) and (32), we get the relation

$$QQ'_1\underline{\theta_1} \diamondsuit_\ell^{\mathsf{dom}(Q)\cup I_2\cup\mathsf{dom}(\theta_1)} QQ'_2\underline{\theta'_2}\underline{\theta_1}$$

(40). Additionally, from (10), we have $I_1 \subseteq \mathsf{dom}(Q'_1/\tau'_1) \cup \mathsf{dom}(\theta_1)$, that is, $I_1 \subseteq I_2 \cup \mathsf{dom}(\theta_1)$. Hence, by Property 3.4.2.i (page 106) applied to (40), we get the relation $QQ'_1\underline{\theta_1} \diamondsuit_\ell^{\mathsf{dom}(Q)\cup I_1} QQ'_2\underline{\theta'_2}\underline{\theta_1}$ **(41)**. Then, by transitivity, (11), and (41), we get $QQ_1 \diamondsuit_\ell^{\mathsf{dom}(Q)\cup I_1} QQ'_2\underline{\theta'_2}\underline{\theta_1}$ **(42)**. Let θ be $\theta'_2 \circ \theta_1$ restricted to I_1, so that $\mathsf{dom}(\theta) \subseteq I_1$ **(43)**

holds. By Eq-Free and (42), we get $QQ_1 \Diamond_\ell^{\mathsf{dom}(Q)\cup I_1} QQ'_2\underline{\theta}$ $\textbf{(\underline{44})}$. We have

$$
\begin{array}{rcll}
\theta(I_1) & = & \theta'_2 \circ \theta_1(I_1) & \text{by definition} \\
& \subseteq & \theta'_2(\mathsf{dom}(Q) \cup \mathsf{dom}(Q'_1/\tau'_1)) & \text{by (10).} \\
& = & \mathsf{dom}(Q) \cup \theta'_2(\mathsf{dom}(Q'_1/\tau'_1)) & \text{since } \mathsf{dom}(\theta'_2) \# \mathsf{dom}(Q) \text{ by (21).} \\
& = & \mathsf{dom}(Q) \cup \theta'_2(I_2) & \text{by (14).} \\
& \subseteq & \mathsf{dom}(Q) \cup \mathsf{dom}(Q'_2/\tau'_2) & \text{by (33).}
\end{array}
$$

Consequently, we have $\theta(I_1) \subseteq \mathsf{dom}(Q) \cup \mathsf{dom}(Q'_2/\tau'_2)$ $\textbf{(\underline{45})}$.

We have $\mathsf{ftv}(\tau_1) \subseteq \mathsf{dom}(Q) \cup \mathsf{dom}(Q_1/\tau_1)$. Hence, $\mathsf{ftv}(\theta_1(\tau_1)) \subseteq \mathsf{dom}(Q) \cup \theta_1(I_1)$, that is, by (10), $\mathsf{ftv}(\theta_1(\tau_1)) \subseteq \mathsf{dom}(Q) \cup I_2$. Additionally, $\mathsf{ftv}(\tau'_1) \subseteq \mathsf{dom}(Q) \cup \mathsf{dom}(Q'_1/\tau'_1)$, that is, $\mathsf{ftv}(\tau'_1) \subseteq \mathsf{dom}(Q) \cup I_2$ by (14). Hence, from (7), (32) and Lemma 3.4.3, we get $(QQ'_2\theta'_2)\ \theta_1(\tau_1) \equiv \tau'_1$. By R-Context-R, we get $(QQ'_2)\ \theta'_2(\theta_1(\tau_1)) \equiv \theta'_2(\tau'_1)$, that is, $(QQ'_2)\ \theta(\tau_1) \equiv \theta'_2(\tau'_1)$. By R-Trans and (22), we get $(QQ'_2)\ \theta(\tau_1) \equiv \tau'_2$ $\textbf{(\underline{46})}$.

We have the expected result, that is, (46), (8), (43), (45) and (44).

∘ Case I-Bot: We have $\sigma_1 = \bot$, thus $\mathsf{nf}(\sigma_1)$ is $\bot$ and this case is not possible by hypothesis.

∘ Case I-Rigid: By taking $\theta = \mathtt{id}$ and by PI-Rigid.

∘ Case R-Context-R: We have $\sigma_1 = \forall\,(\alpha \diamond \sigma)\ \sigma'_1$ and $\sigma_2 = \forall\,(\alpha \diamond \sigma)\ \sigma'_2$. The premise is $(Q, \alpha \diamond \sigma)\ \sigma'_1 \Diamond \sigma'_2$ $\textbf{(\underline{47})}$. We proceed by case analysis:

Subcase $\mathsf{nf}(\sigma'_1) = \alpha$ and $\mathsf{nf}(\sigma'_2) = \alpha$: Hence, $\mathsf{cf}(\sigma_1)$ is $\mathsf{cf}(\sigma)$ and $\mathsf{cf}(\sigma_2)$ is $\mathsf{cf}(\sigma)$, thus we get the expected result by taking $\theta = \mathtt{id}$.

Subcase $\mathsf{nf}(\sigma'_1) \neq \alpha$ and $\mathsf{nf}(\sigma'_2) = \alpha$ is not possible since the derivation is thrifty.

Subcase $\mathsf{nf}(\sigma'_1) = \alpha$ and $\mathsf{nf}(\sigma'_2) \neq \alpha$ is not possible since the derivation is thrifty.

Otherwise $\mathsf{nf}(\sigma'_1) \neq \alpha$ and $\mathsf{nf}(\sigma'_2) \neq \alpha$ $\textbf{(\underline{48})}$: By definition, $\mathsf{cf}(\sigma_1)$ is $\forall\,(\alpha \diamond \sigma)$ $\mathsf{cf}(\sigma'_1)$ and $\mathsf{cf}(\sigma_2)$ is $\forall\,(\alpha \diamond \sigma)\ \mathsf{cf}(\sigma'_2)$. If we have $\mathsf{nf}(\sigma'_1) = \bot$, then $\mathsf{nf}(\sigma_1)$ is $\bot$, which is not possible by hypothesis. Hence, $\mathsf{nf}(\sigma'_1) \neq \bot$. If we have $\mathsf{nf}(\sigma'_2) \in \mathsf{dom}(Q, \alpha \diamond \sigma)$, and $(Q, \alpha \diamond \sigma)(\mathsf{nf}(\sigma'_2)) \notin \mathcal{T}$, then, either $\mathsf{nf}(\sigma'_2) \in \mathsf{dom}(Q)$ $\textbf{(\underline{49})}$ and $Q(\mathsf{nf}(\sigma'_2)) \notin \mathcal{T}$ $\textbf{(\underline{50})}$, or $\mathsf{nf}(\sigma'_2)$ is α. The latter is not possible by (48). In the former case, we have $\mathsf{nf}(\sigma_2) = \mathsf{nf}(\sigma'_2)$, thus (49) and (50) give $\mathsf{nf}(\sigma_2) \in \mathsf{dom}(Q)$ and $Q(\mathsf{nf}(\sigma_2)) \notin \mathcal{T}$, which is not possible by hypothesis. Hence, $\mathsf{nf}(\sigma'_2) \notin \mathsf{dom}((Q, \alpha \diamond \sigma))$, or $(Q, \alpha \diamond \sigma)(\mathsf{nf}(\sigma'_2)) \in \mathcal{T}$. We have $\mathsf{cf}(\sigma_1) = \forall\,(\alpha \diamond \sigma)\ \forall\,(Q'_1)\ \tau_1$. By induction hypothesis on (47), there exists an alpha-conversion of $\mathsf{cf}(\sigma'_2)$ written $\forall\,(Q'_2)\ \tau_2$ and a substitution θ such that

$$
(Q, \alpha \diamond \sigma, Q'_2)\ \theta(\tau_1) \equiv \tau_2 \qquad I \triangleq \mathsf{dom}(Q'_1/\tau_1) \qquad \mathsf{dom}(\theta) \subseteq I
$$

$$
\theta(I) \subseteq \mathsf{dom}(Q) \cup \{\alpha\} \cup \mathsf{dom}(Q'_2/\tau_2) \qquad (Q, \alpha \diamond \sigma, Q'_1) \Diamond_\ell^{\mathsf{dom}(Q)\cup\{\alpha\}\cup I} (Q, \alpha \diamond \sigma, Q'_2)\underline{\theta}\ \textbf{(\underline{51})}
$$

Let Q_1 be $(\alpha \diamond \sigma, Q'_1)$ and Q_2 be $(\alpha \diamond \sigma, Q'_2)$. The constructed form of σ_1 is $\forall\,(Q_1)\ \tau_1$ and $\forall\,(Q_2)\ \tau_2$ is an alpha-conversion of $\mathsf{cf}(\sigma_2)$. By defining $I' \triangleq \mathsf{dom}(Q_1/\tau_1)$, we have $I \subseteq I' \subseteq I \cup \{\alpha\}$ **(<u>52</u>)**. We immediately have

$$(QQ_2)\ \theta(\tau_1) \equiv \tau_2 \qquad I' \triangleq \mathsf{dom}(Q_1/\tau_1) \qquad \mathsf{dom}(\theta) \subseteq I'$$

$$\theta(I') \subseteq \mathsf{dom}(Q) \cup \mathsf{dom}(Q_2/\tau_2) \qquad (QQ_1)\ \Diamond_{\ell}^{\mathsf{dom}(Q) \cup I'} (QQ_2)\underline{\theta}\ (\underline{\mathbf{53}})$$

We get (53) from (51) by using Property 3.4.2.i (page 106) and (52). ■

Proof of Lemma 3.4.7

<u>Directly</u>, $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ holds by hypothesis, and Q is unconstrained. Besides, σ_1 and σ_2 are ML types. We proceed by case analysis:

∘ Case $\mathsf{nf}(\sigma_1)$ is $\bot$: Then σ_1 is necessarily of the form $\forall\,(Q_1)\ \alpha_1$, with $\alpha_1 \in \mathsf{dom}(Q_1)$ and Q_1 is an unconstrained prefix. In ML, σ_1 is written $\forall\,(\bar\alpha)\ \forall\,(\alpha)\ \alpha$, and we have $\sigma_1 \sqsubseteq_{ML} \sigma_2$, provided σ_2 is an ML type.

∘ Case σ_1 is in $\mathcal{V}$: We have $\mathsf{nf}(\sigma_1) = \alpha$. Necessarily, σ_1 is of the form $\forall\,(\bar\alpha)\ \alpha$ with $\alpha \notin \bar\alpha$. By Lemma 2.1.6, $(Q)\ \sigma_2 \equiv \alpha$ holds. Thus $\widehat{Q}(\alpha)$ is a rearrangement of $\widehat{Q}(\mathsf{nf}(\sigma_2))$ by Lemma 1.5.9. Observing that $\widehat{Q} = \mathtt{id}$, we get $\mathsf{nf}(\sigma_2) = \alpha$. Hence, σ_2 is of the form $\forall\,(\bar\beta)\ \alpha$ with $\alpha \notin \bar\beta$. Consequently, σ_1 and σ_2 are equivalent in ML.

∘ Case σ_2 is in $\mathcal{V}$: We assume $\mathsf{nf}(\sigma_1)$ is not $\bot$ and σ_1 is not in $\mathcal{V}$. By Properties 2.1.5.ii and 2.1.5.iii (page 67), σ_1/ϵ is a type constructor g. Hence, $(\forall\,(Q)\ \sigma_1)/\epsilon$ is g. Since Q is unconstrained and $\sigma_2 \in \mathcal{V}$, we have $(\forall\,(Q)\ \sigma_2)/\epsilon = \bot$. This is a contradiction by Property 2.1.3.ii (page 65). Hence, this case cannot occur.

∘ Case σ_2 is not in constructed form: Necessarily, σ_2 is of the form $\forall\,(\bar\alpha)\ \alpha$, with $\alpha \in \bar\alpha$. Hence, σ_2/ϵ is $\bot$. By Property 2.1.3.ii (page 65), we must have $\sigma_1/\epsilon = \bot$ or $\sigma_1/\epsilon = \alpha$ with $\alpha \in \mathsf{dom}(Q)$. In the first case, $\mathsf{nf}(\sigma_1)$ is $\bot$ by Property 2.1.5.iii (page 67) and Property 1.5.11.i (page 54), thus this case has already been solved above. In the second case, $\sigma_1 \equiv \alpha$ by Property 2.1.5.ii (page 67), thus $\sigma_1 \in \mathcal{T}$ by Property 1.5.11.x (page 54) and $(Q)\ \sigma_1 \equiv \sigma_2$ holds by Lemma 2.1.6. Thus, $\widehat{Q}(\sigma_1) \equiv \widehat{Q}(\sigma_2)$ holds by Corollary 1.5.10. Observing that $\widehat{Q}$ is $\mathtt{id}$, we get $\sigma_1 \equiv \sigma_2$, which is a contradiction with Property 1.5.4.i (page 50).

∘ Otherwise: σ_1 is of the (constructed) form $\forall\,(Q_1)\ \tau_1$, where Q_1 is unconstrained. Besides, σ_2 is in constructed form too. By Lemma 3.4.4, there exists an alpha-conversion of σ_2, written $\forall\,(Q_2)\ \tau_2$, and a substitution θ such that $(QQ_2)\ \theta(\tau_1) \equiv \tau_2$ holds, with $\mathsf{dom}(\theta) \subseteq \mathsf{dom}(Q_1)$. Observing that QQ_2 is unconstrained, we get $\theta(\tau_1) = \tau_2$ by Property 1.5.11.vii (page 54). Let $\bar\alpha$ be $\mathsf{dom}(Q_1)$ and $\bar\beta$ be $\mathsf{dom}(Q_2)$. We can write θ in the form $[\bar\tau/\bar\alpha]$. Hence, σ_2 is $\forall\,(\bar\beta)\ \tau_1[\bar\tau/\bar\alpha]$ (up to renaming) As a consequence, σ_2 is an instance of σ_1 in ML.

Conversely, σ_2 is by hypothesis an instance of σ_1. This means that σ_1 is $\forall\,(\bar{\alpha})\ \tau$ and σ_2 is $\forall\,(\bar{\beta})\ \tau[\bar{\tau'}/\bar{\alpha}]$. We can derive $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ by rules R-Context-Flexible, I-Bot and Eq-Mono*. ■

Proof of Lemma 3.5.2

By induction on the size of Q. If Q is $\emptyset$, then necessarily $\bar{\alpha} = \emptyset$, and the algorithm returns the pair $(\emptyset, \emptyset)$, which is correct. Otherwise, Q is of the form $Q'q$ **(1)**, where q is $(\alpha \diamond \sigma)$. We consider two cases:

- Case $\alpha \notin \bar{\alpha}$: Let (Q'_1, Q'_2) be $Q' \uparrow \bar{\alpha}$. By definition, we have $Q_1 = Q'_1$ **(2)** and $Q_2 = (Q'_2 q)$ **(3)**. By induction hypothesis, we have $Q'_1 Q'_2 \approx Q'$ **(4)**, $\bar{\alpha} \subseteq \mathsf{dom}(Q'_1)$ **(5)**, as well as $\mathsf{dom}(Q'_1/\bar{\alpha}) = \mathsf{dom}(Q'_1)$ **(6)**. Then $Q'_1 Q'_2 q \approx Q'q$ holds from (4), (1), (2), and (3), that is, $Q_1 Q_2 \approx Q$ (i). By (2), (5) and (6), we have $\bar{\alpha} \subseteq \mathsf{dom}(Q_1)$ and $\mathsf{dom}(Q_1/\bar{\alpha}) = \mathsf{dom}(Q_1)$, which is the expected result (ii) and (iii).

- Case $\alpha \in \bar{\alpha}$: Let $\bar{\beta}$ be $(\bar{\alpha} - \alpha) \cup \mathsf{ftv}(\sigma)$. By definition 3.3.1 and (1), we have $\mathsf{dom}(Q/\bar{\alpha}) = \alpha \cup \mathsf{dom}(Q'/\bar{\beta})$ **(7)**. Let (Q'_1, Q'_2) be $Q' \uparrow \bar{\beta}$. By definition, we have $Q_1 = Q'_1 q$ **(8)** and $Q_2 = Q'_2$ **(9)**. By induction hypothesis, $Q'_1 Q'_2 \approx Q'$ **(10)**, $\bar{\beta} \subseteq \mathsf{dom}(Q'_1)$ **(11)** and $\mathsf{dom}(Q'_1/\bar{\beta}) = \mathsf{dom}(Q'_1)$ **(12)**. We have $\mathsf{ftv}(\sigma) \subseteq \bar{\beta}$. Hence, by (11), $\mathsf{ftv}(\sigma) \subseteq \mathsf{dom}(Q'_1)$, which implies $\mathsf{ftv}(\sigma)\ \#\ \mathsf{dom}(Q'_2)$. Hence, we have $Q'_1 q Q'_2 \approx Q'_1 Q'_2 q$, thus $Q'_1 q Q'_2 \approx Q'q$ holds from (10), that is, $Q_1 Q_2 \approx Q$ (i) from (8) and (9). From (11), we have $\bar{\alpha} \subseteq \mathsf{dom}(Q'_1) \cup \{\alpha\}$, that is, $\bar{\alpha} \subseteq \mathsf{dom}(Q_1)$ (ii). Finally, $\mathsf{dom}(Q/\bar{\alpha})$ is $\mathsf{dom}(Q'_1 q Q'_2/\bar{\alpha})$ **(13)** by Property 3.3.2.ii (page 104) and (i). Besides, $\mathsf{dom}(Q'_1 q Q'_2/\bar{\alpha})$ is $\mathsf{dom}(Q'_1 q/\bar{\alpha})$ **(14)** since $\bar{\alpha}\ \#\ \mathsf{dom}(Q'_2)$. Similarly, $\mathsf{dom}(Q'/\bar{\beta})$ is $\mathsf{dom}(Q'_1/\bar{\beta})$ **(15)**.

  Thus, we can prove the following sequence of equalities:

$$\begin{array}{rcll}
\mathsf{dom}(Q_1/\bar{\alpha}) & = & \mathsf{dom}(Q'_1 q/\bar{\alpha}) & \text{by (8)} \\
 & = & \mathsf{dom}(Q'_1 q Q'_2/\bar{\alpha}) & \text{by (14)} \\
 & = & \mathsf{dom}(Q/\bar{\alpha}) & \text{by (13)} \\
 & = & \alpha \cup \mathsf{dom}(Q'/\bar{\beta}) & \text{by (7)} \\
 & = & \alpha \cup \mathsf{dom}(Q'_1/\bar{\beta}) & \text{by (15)} \\
 & = & \alpha \cup \mathsf{dom}(Q'_1) & \text{by (12)} \\
 & = & \mathsf{dom}(Q_1) & \text{by (8)}
\end{array}$$

This is the expected result (iii). ■

Proof of Lemma 3.6.2

By hypothesis, we have $Q \diamondsuit^I Q'$ **(1)**. Let (Q_1, Q'_0) be $Q \uparrow I$. By Lemma 3.5.2, $Q_1 Q'_0$ **(2)** is a rearrangement of Q, $I \subseteq \mathsf{dom}(Q_1)$ **(3)** and $\mathsf{dom}(Q_1/I) = \mathsf{dom}(Q_1)$ **(4)**. We get $\mathsf{dom}(Q/I) = \mathsf{dom}(Q_1)$ **(5)** from (4) and (2). From (2), we get $\mathsf{dom}(Q'_0)\ \#\ \mathsf{dom}(Q_1)$, which implies $\mathsf{dom}(Q'_0) \# \mathsf{dom}(Q/I)$ $\mathbf{(\underline{6})}$ by (5) as well as $\mathsf{dom}(Q'_0) \# I$ $\mathbf{(\underline{7})}$ by (3). We have $Q \equiv^I Q_1$ by (2), Property 3.2.2.ii (page 103), PE-Free, and (7). Thus $Q_1 \diamondsuit^I Q'$ $\mathbf{(\underline{8})}$ holds from (1) and R-Trans. Moreover, $\mathsf{dom}(Q/I) \# \mathsf{dom}(Q'_0)$ holds from (2) and (5). Let ϕ be a renaming of $\mathsf{dom}(Q')$ disjoint from I such that $\mathsf{dom}(\phi(Q')) \cap \mathsf{dom}(Q_1) = I$ and $\phi(Q') \# Q'_0$ $\mathbf{(\underline{9})}$. By Lemma 3.6.1 (page 113) and (8), there exists a substitution θ such that $Q_1 \diamondsuit^J \phi(Q')\underline{\theta}$ $\mathbf{(\underline{10})}$ holds, where J is $\mathsf{dom}(Q_1/I)$, that is $\mathsf{dom}(Q_1)$ $\mathbf{(\underline{11})}$ by (4). Additionally, we have $\mathsf{dom}(\theta) \subseteq J - I$, which implies $\mathsf{dom}(\theta) \# I$ $\mathbf{(\underline{12})}$ and $\mathsf{dom}(\theta) \# \mathsf{dom}(Q'_0)$ $\mathbf{(\underline{13})}$ by (11) and (2). We have $Q_1 \diamondsuit \phi(Q')\underline{\theta}$ $\mathbf{(\underline{14})}$ from (10) and (11). By Property 3.4.2.iii (page 106), (9), (13), and (14) we get $Q_1Q'_0 \diamondsuit \phi(Q')\underline{\theta}Q'_0$ $\mathbf{(\underline{15})}$. We define Q_0 as $\underline{\theta}Q'_0$ $\mathbf{(\underline{16})}$. Note that $\mathsf{dom}(Q_0) \# I$ $\mathbf{(\underline{17})}$ by (7) and (12). From (15), (2), and (16), we get $Q \diamondsuit \phi(Q')Q_0$ $\mathbf{(\underline{18})}$. Finally, we have $Q' \equiv \phi(Q')\underline{\phi}$ by Property 3.4.2.iv (page 106). Since ϕ is disjoint from I, we get $Q' \equiv^I \phi(Q')$ $\mathbf{(\underline{19})}$ by PE-Free. Besides we have $\phi(Q')Q_0 \equiv^I \phi(Q')$ $\mathbf{(\underline{20})}$ by PE-Free and (17). Hence, by PE-Trans, (19), and (20), we get $Q' \equiv^I \phi(Q')Q_0$ $\mathbf{(\underline{21})}$. We have shown the expected results, namely, (18), (17), (21), (16), and (6). ■

Proof of Property 3.6.3

We prove each property independently.

Property i: By hypothesis, $(Q, \alpha \diamond \sigma)\ \sigma_1 \diamondsuit \sigma_2$ holds. Hence, $(Q)\ \forall (\alpha \diamond \sigma)\ \sigma_1 \diamondsuit \forall (\alpha \diamond \sigma)\ \sigma_2$ holds by R-Context-R. We conclude by R-Trans and observing that $(Q)\ \forall (\alpha \diamond \sigma)\ \sigma_1 \equiv \sigma_1$ and $(Q)\ \forall (\alpha \diamond \sigma)\ \sigma_2 \equiv \sigma_2$ hold by Eq-Free.

Property ii: We prove by induction that for any Q, Q_0, σ_1 and σ_2, if $(Q, \alpha \geq \tau, Q_0)\ \sigma_1 \diamondsuit \sigma_2$ holds, then $(Q, \alpha = \tau, Q_0)\ \sigma_1 \diamondsuit \sigma_2$ holds too. Equivalence cases are discarded since $\diamondsuit$ is $\mathrel{-\!\!\!\!\sqsubseteq}$ or $\sqsubseteq$. Cases R-Trans, R-Context-Rigid, R-Context-Flexible and I-Abstract are by induction hypothesis. Cases I-Bot and I-Rigid do not read the prefix, thus $(Q, \alpha = \tau, Q_0)\ \sigma_1 \sqsubseteq \sigma_2$ still holds.

- Case R-Context-R we have $\sigma_1 = \forall (\beta \diamond \sigma)\ \sigma'_1$ and $\sigma_2 = \forall (\beta \diamond \sigma)\ \sigma'_2$. By hypothesis, $(Q, \alpha \geq \tau, Q_0, \beta \diamond \sigma)\ \sigma'_1 \diamondsuit \sigma'_2$ holds. By induction hypothesis, $(Q, \alpha = \tau, Q_0, \beta \diamond \sigma)\ \sigma'_1 \diamondsuit \sigma'_2$ holds too. Hence, $(Q, \alpha = \tau, Q_0)\ \sigma_1 \diamondsuit \sigma_2$ holds by R-Context-R.
- Case A-Hyp: Then $\sigma_2 = \beta$. If β is not α, the result is immediate. Otherwise, $(Q, \alpha \diamond \tau, Q_0)\ \tau \mathrel{-\!\!\!\!\sqsubseteq} \alpha$ holds by Eq-Mono.
- Case I-Hyp: similar.

Remark that we do not read the prefix, except in the last two cases. As a consequence, we use the same proof structure for the next properties, and we only need to show the result for cases A-Hyp and I-Hyp.

Property iii: We have $(Q, \alpha \diamond \sigma, \alpha' = \alpha, Q_0)\ \sigma_1 \diamondsuit \sigma_2$.

∘ Case A-Hyp: We have $\sigma_2 = \beta$. If β is neither α nor α', we get the result by A-Hyp. If β is α, we have $\sigma_1 = \sigma$, and $(Q, \alpha' \diamond \sigma, \alpha = \alpha', Q_0)\ \sigma_1 \sqsubseteq \alpha'$ holds by A-Hyp, thus $(Q, \alpha' \diamond \sigma, \alpha = \alpha', Q_0)\ \sigma_1 \sqsubseteq \alpha$ holds by R-Trans, Eq-Mono and A-Equiv. If β is α', we have $\sigma_1 = \alpha$, and $(Q, \alpha' \diamond \sigma, \alpha = \alpha', Q_0)\ \sigma_1 \sqsubseteq \alpha'$ holds by Eq-Mono.

∘ Case I-Hyp: similar.

Property iv: We have $(Q)\ \sigma \equiv \sigma'$ (**1**) and $(Q, \alpha \diamond \sigma, Q_0)\ \sigma_1 \diamondsuit \sigma_2$. As explained above, we only need to consider cases A-Hyp and I-Hyp. In both cases, σ_2 is a variable β.

∘ Case A-Hyp: If β is not α, the result is by A-Hyp. Otherwise, we have $\sigma_1 = \sigma$ and $\diamond$ is $=$. We have $(Q, \alpha \diamond \sigma', Q_0)\ \sigma \equiv \sigma'$ (**2**) from (1) and Property 1.5.3.v (page 49). Additionally, $(Q, \alpha \diamond \sigma', Q_0)\ \sigma' \sqsubseteq \alpha$ (**3**) holds by A-Hyp. Hence, $(Q, \alpha \diamond \sigma', Q_0)\ \sigma \sqsubseteq \alpha$ holds by A-Equiv, R-Trans, (2), and (3).

∘ Case I-Hyp: similar.

Property v: We have $(Q)\ \sigma \sqsubseteq \sigma'$ (**1**) and $(Q, \alpha = \sigma, Q_0)\ \sigma_1 \diamondsuit \sigma_2$.

∘ Case A-Hyp: We have $\sigma_2 = \beta$. If β is not α, we get the result by A-Hyp. Otherwise, σ_1 is σ. By Property 1.7.2.iii (page 59), $(Q, \alpha = \sigma', Q_0)\ \sigma \sqsubseteq \sigma'$ (**2**) holds from (1). Hence, $(Q, \alpha = \sigma', Q_0)\ \sigma \sqsubseteq \alpha$ holds by R-Trans, (2), and A-Hyp.

∘ Case I-Hyp: We have $\sigma_2 = \beta$. We cannot have $\beta = \alpha$, since the binding of α is rigid. Thus, we get the result by I-Hyp.

Property vi: We have $(Q)\ \sigma \sqsubseteq \sigma'$ (**1**) and $(Q, \alpha \geq \sigma, Q_0)\ \sigma_1 \diamondsuit \sigma_2$.

∘ Case A-Hyp: We have $\sigma_2 = \beta$. We cannot have $\beta = \alpha$, since the binding of α is flexible. Thus, we get the result by A-Hyp.

∘ Case I-Hyp: We have $\sigma_2 = \beta$. If β is not α, we get the result by I-Hyp. Otherwise, σ_1 is σ. By Property 1.7.2.iii (page 59) and (1), $(Q, \alpha \geq \sigma', Q_0)\ \sigma \sqsubseteq \sigma'$ (**2**) holds. Hence, $(Q, \alpha \geq \sigma', Q_0)\ \sigma \sqsubseteq \alpha$ holds by R-Trans, (2), and I-Hyp.

Property vii: We have $(Q, \alpha \geq \sigma, Q_0)\ \sigma_1 \diamondsuit \sigma_2$.

∘ Case A-Hyp: We have $\sigma_2 = \beta$. We cannot have $\beta = \alpha$, since the binding of α is flexible. Thus, we get the result by A-Hyp.

∘ Case I-Hyp: We have $\sigma_2 = \beta$. If β is not α, we get the result by I-Hyp. Otherwise, σ_1 is σ. By A-Hyp, we can derive $(Q, \alpha = \sigma, Q_0)\ \sigma_1 \sqsubseteq \alpha$. Then by I-Abstract, we get $(Q, \alpha = \sigma, Q_0)\ \sigma_1 \sqsubseteq \alpha$. This is the expected result. ■

Proof of Lemma 3.6.4

If $\diamondsuit$ is $\equiv$, then we get the result by Lemma 3.4.3. Otherwise, $\diamondsuit$ is $\mathrel{\sqsubset\!\!\!\!-}$ or $\sqsubseteq$. The proof is by induction on the derivation of $Q_1 \diamondsuit^I Q_2$. Cases PE-Trans, PA-Trans, PI-Trans, PA-Equiv, PI-Abstract are by induction hypothesis. Case PE-Refl is immediate. Other cases are:

∘ Case PE-Free: If Q_1 is $(Q_2, \alpha \diamond \sigma)$, we get the result by Property 3.6.3.i (page 114). If Q_2 is $(Q_1, \alpha \diamond \sigma)$, we get the result by Property 1.7.2.iii (page 59).

∘ Case PE-Mono: Direct consequence of Property 3.6.3.ii (page 114).

∘ Case PE-Context-L: Direct consequence of Property 3.6.3.iv (page 114).

∘ Case PE-Swap: Direct consequence of Property 3.6.3.iii (page 114).

∘ Case PE-Comm: Direct consequence of Property 1.7.2.ii (page 59).

∘ Case PA-Context-L: Direct consequence of Property 3.6.3.v (page 114).

∘ Case PI-Context-L: Direct consequence of Property 3.6.3.vi (page 114).

∘ Case PI-Rigid: Direct consequence of Property 3.6.3.vii (page 114). ■

Proof of Lemma 3.6.6

We have the hypotheses

$$(Q)\ \sigma_1 \sqsubseteq \sigma_2\ (\underline{\mathbf{1}}) \qquad (Q)\ \sigma_2 \sqsubseteq \sigma_3\ (\underline{\mathbf{2}}) \qquad (Q)\ \sigma_1 \mathrel{\sqsubset\!\!\!\!-} \sigma_3\ (\underline{\mathbf{3}})$$

We consider two cases:

∘ Case $\widehat{Q}(\mathsf{nf}(\sigma_3)) \notin \mathcal{V}$: Then we have $\widehat{Q}(\mathsf{nf}(\sigma_2)) \notin \mathcal{V}$ by Lemmas 2.1.6, (2), and 1.5.9. We have $\forall\,(Q)\ \sigma_1/ \leqslant_{/} \forall\,(Q)\ \sigma_2/ \leqslant_{/} \forall\,(Q)\ \sigma_3/$ by Property 2.1.3.ii (page 65), (1), and (2). We have $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_3/$ by Property 2.1.3.i (page 65) and (3). Hence, by antisymmetry (Property 2.1.2.i (page 65)), we get $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/ = \forall\,(Q)\ \sigma_3/$. We have $w(\sigma_1) \geqslant w(\sigma_2) \geqslant w(\sigma_3)$ by Property 2.7.6.i (page 94), (1), and (2). We have $X \notin w(\sigma_1) - w(\sigma_3)$ by Lemma 2.7.8 and (3), thus we get $X \notin w(\sigma_1) - w(\sigma_2)$ and $X \notin w(\sigma_2) - w(\sigma_3)$ by Property 2.7.2.iv (page 87). Hence, by Lemma 2.7.8 and (1), we get $(Q)\ \sigma_1 \mathrel{\sqsubset\!\!\!\!-} \sigma_2$. Similarly, by Lemma 2.7.8 and (2), we get $(Q)\ \sigma_2 \mathrel{\sqsubset\!\!\!\!-} \sigma_3$.

∘ Case $\widehat{Q}(\mathsf{nf}(\sigma_3)) = \alpha$ $(\underline{\mathbf{4}})$: Note that necessarily $\widehat{Q}(\alpha) = \alpha$ $(\underline{\mathbf{5}})$ holds, because $\widehat{Q}$ is idempotent. We consider two subcases.

Subcase $\sigma_2 \in \mathcal{V}$: Then $(Q)\ \sigma_2 \equiv \sigma_3$ $(\underline{\mathbf{6}})$ holds by Lemma 2.1.6, thus $(Q)\ \sigma_1 \mathrel{\sqsubset\!\!\!\!-} \sigma_2$ holds by A-Equiv, R-Trans, (6) and (3). Besides, $(Q)\ \sigma_2 \mathrel{\sqsubset\!\!\!\!-} \sigma_3$ holds by A-Equiv and (6).

Subcase $\mathsf{nf}(\sigma_2) \notin \vartheta$: Then we have $\sigma_2 \notin \mathcal{V}$ $(\underline{\mathbf{7}})$. If we have $\sigma_1 \in \mathcal{V}$, then by (1), (2), and Lemma 2.1.6, we get $(Q)\ \sigma_1 \equiv \sigma_2$ and $(Q)\ \sigma_2 \equiv \sigma_3$, which leads to the expected result by A-Equiv. Now, we assume $\sigma_1 \notin \mathcal{V}$ $(\underline{\mathbf{8}})$. From (2), we get $(Q)\ \sigma_2 \sqsubseteq \widehat{Q}(\mathsf{nf}(\sigma_3))$ by Eq-Mono, Property 1.5.6.i (page 51), and I-Equiv*.

Hence, we have $(Q)\ \sigma_2 \sqsubseteq \alpha$ $\mathbf{(\underline{9})}$ from (4). Let $(\alpha \diamond \sigma)$ be the binding of α in Q $\mathbf{(\underline{10})}$. We get by Corollary 2.3.4, (10), (7), and (9) $(Q)\ \sigma_2 \sqsubseteq \sigma$ $\mathbf{(\underline{11})}$. From (4), we have $(Q)\ \sigma_3 \equiv \alpha$ $\mathbf{(\underline{12})}$ by Eq-Mono and Property 1.5.6.i. Hence, by (3) and R-Trans, we get $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \alpha$ $\mathbf{(\underline{13})}$. By Corollary 2.3.4, (10), (8), and (13), we get $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma$ $\mathbf{(\underline{14})}$ and $\diamond$ is rigid, that is, $(\alpha{=}\sigma) \in Q$ $\mathbf{(\underline{15})}$. We have $\widehat{Q}(\mathsf{nf}(\sigma)) \notin \mathcal{V}$ $\mathbf{(\underline{16})}$ by (15) and (5). In summary, we have shown $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ from (1), $(Q)\ \sigma_2 \sqsubseteq \sigma$ from (11), and $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma$ from (14). Hence, thanks to (16), we fall back in the first case. As a consequence, we have $(Q)\ \sigma_1 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma_2$ $\mathbf{(\underline{17})}$ and $(Q)\ \sigma_2 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma$ $\mathbf{(\underline{18})}$.

By A-Hyp and (15), we get $(Q)\ \sigma \mathrel{\sqsubseteq\!\!\!\!\!-} \alpha$ $\mathbf{(\underline{19})}$. By R-Trans, (18), and (19), we get $(Q)\ \sigma_2 \mathrel{\sqsubseteq\!\!\!\!\!-} \alpha$ $\mathbf{(\underline{20})}$. By R-Trans, (20) and (12), we get $(Q)\ \sigma_2 \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma_3$. With (17), this is the expected result. ■

Proof of Lemma 3.6.7

By hypothesis, we have $(Q)\ \forall(\alpha)\ \sigma \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma'$, where Q is unconstrained and binds the free variables of $\forall(\alpha)\ \sigma$ and σ'. By Property 2.6.2.i (page 85), there exists a derivation of $(Q)\ \forall(\alpha)\ \sigma\ (\equiv\dot{\mathrel{\sqsubseteq\!\!\!\!\!-}}^{\emptyset})^{*}\ \sigma'$. More precisely, there exist $\sigma_1, \ldots \sigma_n$, such that $\sigma_1 = \forall(\alpha)\ \sigma$, $\sigma_n = \sigma'$, and $(Q)\ \sigma_i\ (\equiv\dot{\mathrel{\sqsubseteq\!\!\!\!\!-}}^{\emptyset})\ \sigma_{i+1}$ for $i \in [1..n[$. We prove the following property (where Q is unconstrained):

> *If we have $\sigma_a \equiv \forall(\alpha)\ \sigma'_a$ and $(Q)\ \sigma_a\ (\equiv\dot{\mathrel{\sqsubseteq\!\!\!\!\!-}}^{\emptyset})\ \sigma_b$, then there exists σ'_b such that $\sigma_b \equiv \forall(\alpha)\ \sigma'_b$ and $(Q, \alpha)\ \sigma'_a \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma'_b$.*

Proof: By hypothesis, we have $\sigma_a \equiv \forall(\alpha)\ \sigma'_a$ $\mathbf{(\underline{1})}$ and $(Q)\ \sigma_a\ (\equiv\dot{\mathrel{\sqsubseteq\!\!\!\!\!-}}^{\emptyset})\ \sigma_b$. By definition, either $\sigma_a \equiv \sigma_b$ (the result is immediate, then), or there exist σ_c and σ_d such that we have

$$\sigma_a \equiv \sigma_c\ (\underline{\mathbf{2}}) \qquad\qquad (Q)\ \sigma_c \dot{\mathrel{\sqsubseteq\!\!\!\!\!-}}^{\bar{\alpha}} \sigma_d\ (\underline{\mathbf{3}}) \qquad\qquad \sigma_d \equiv \sigma_b\ (\underline{\mathbf{4}}).$$

Hence, we have $\sigma_c \equiv \forall(\alpha)\ \sigma'_a$ $\mathbf{(\underline{5})}$ by R-Trans, (1) and (2). We can choose α such that $\alpha \notin \mathsf{ftv}(\sigma_b)$. If $\alpha \notin \mathsf{ftv}(\sigma'_a)$, then $\sigma_c \equiv \sigma'_a$, thus taking $\sigma'_b = \sigma_b$ is appropriate. Hence, we assume that $\alpha \in \mathsf{ftv}(\sigma'_a)$ $\mathbf{(\underline{6})}$. By Property 2.6.3.i (page 86) on (3), there exists σ'_d such that $(Q)\ \mathsf{nf}(\sigma_c) \dot{\mathrel{\sqsubseteq\!\!\!\!\!-}}^{\bar{\alpha}} \sigma'_d$ $\mathbf{(\underline{7})}$ and $\sigma'_d \equiv \sigma_d$ $\mathbf{(\underline{8})}$. By Property 1.5.11.i (page 54) on (5), $\mathsf{nf}(\sigma_c)$ is a rearrangement of $\mathsf{nf}(\forall(\alpha)\ \sigma'_a)$. By Property 2.6.3.ii (page 86) applied to (7), there exists σ''_d such that we have $(Q)\ \mathsf{nf}(\forall(\alpha)\ \sigma'_a) \dot{\mathrel{\sqsubseteq\!\!\!\!\!-}}^{\bar{\alpha}} \sigma''_d$ $\mathbf{(\underline{9})}$ and $\sigma''_d \equiv \sigma'_d$ $\mathbf{(\underline{10})}$. By R-Trans, (10), (8), and (4), we get $\sigma''_d \equiv \sigma_b$ $\mathbf{(\underline{11})}$. If $\sigma'_a \equiv \alpha$, then $\mathsf{nf}(\forall(\alpha)\ \sigma'_a)$ is $\perp$. This is a contradiction with (9) since no rule can be used to derive $(Q)\ \perp \dot{\mathrel{\sqsubseteq\!\!\!\!\!-}}^{\bar{\alpha}} \sigma''_d$ (note that the prefix is unconstrained). Hence, σ'_a is not equivalent to α. From (6), we get $\mathsf{nf}(\forall(\alpha)\ \sigma'_a) = \forall(\alpha)\ \mathsf{nf}(\sigma'_a)$. Hence, $(Q)\ \forall(\alpha)\ \mathsf{nf}(\sigma'_a) \dot{\mathrel{\sqsubseteq\!\!\!\!\!-}}^{\bar{\alpha}} \sigma''_d$ $\mathbf{(\underline{12})}$ holds from (9). Moreover, (12) is derived by StSh-Up or StSh-Alias (indeed, the prefix is unconstrained). Hence, $\forall(\alpha)\ \mathsf{nf}(\sigma'_a)$ is of the form $C_r(\sigma_0)$ and σ''_d is of the form $C_r(\sigma'_0)$.

Necessarily, C_r is $\forall\,(\alpha)\ C'_r$ and $C'_r(\sigma_0) = \mathsf{nf}(\sigma'_a)$ **(13)**. The premises of StSh-Up or StSh-Alias are independent of C_r and Q. Hence, $(Q,\alpha)\ C'_r(\sigma_0) \mathrel{\dot{-\!\!\!\!\sqsubseteq}}^{\bar\alpha} C'_r(\sigma'_0)$ **(14)** holds too. Let σ'_b be $C'_r(\sigma'_0)$. Then σ''_d is $\forall\,(\alpha)\ \sigma'_b$ **(15)** and (14) gives $(Q,\alpha)\ C'_r(\sigma_0) \mathrel{\dot{-\!\!\!\!\sqsubseteq}}^{\bar\alpha} \sigma'_b$. By Property 2.6.2.i (page 85), we get $(Q,\alpha)\ C'_r(\sigma_0) \mathrel{-\!\!\!\!\sqsubseteq} \sigma'_b$ **(16)**. From (16), (13), and Property 1.5.6.i (page 51), we get $(Q,\alpha)\ \sigma'_a \mathrel{-\!\!\!\!\sqsubseteq} \sigma'_b$. From (11) and (15), we get $\sigma_b \equiv \forall\,(\alpha)\ \sigma'_b$. This is the expected result. ☐

The lemma is then proved by induction on i. ■

Proof of Lemma 3.6.8

By induction on the size of Q. By hypothesis, Q is of the form $(Q_1, \alpha \diamond \sigma_a, Q_2)$. If σ_a/u is defined and is $\bot$, then we get the expected result by taking $\beta = \alpha$, $u_1 = \epsilon$, and $u_2 = u$. Otherwise, there exist v_1 and v_2 such that $u = v_1v_2$ and $\sigma_a/v_1 = \beta$ **(1)** with $\beta \in \mathsf{dom}(Q_1)$. Then $\forall\,(Q_1)\ \beta/v_2 = \bot$. By induction hypothesis, Q_1 is of the form $(Q_{11}, \gamma \diamond \sigma_c, Q_{12})$, v_2 is of the form w_1w_2 such that $\forall\,(Q_{12})\ \beta/w_1 = \gamma$ **(2)** and $\sigma_c/w_2 = \bot$ **(3)**. Then Q is of the form $(Q_{11}, \gamma \diamond \sigma_c, Q_{12}, \alpha \diamond \sigma_a, Q_2)$, u is of the form $v_1w_1w_2$ such that

$$\begin{array}{rcll}
\forall\,(Q_{12}, \alpha \diamond \sigma_a, Q_2)\ \alpha/v_1w_1 & = & \forall\,(Q_{12})\ \sigma_a/v_1w_1 & \\
& = & \forall\,(Q_{12})\ \beta/w_1 & \text{from (1)} \\
& = & \gamma & \text{from (2)}
\end{array}$$

Additionally, $\sigma_c/w_2 = \bot$ holds from (3). This is the expected result. ■

Proof of Lemma 3.6.9

By hypothesis, we have

$$\sigma_2 \in \Sigma_I\ \mathbf{(1)} \qquad \sigma_2 \notin \mathcal{V}\ \mathbf{(2)} \qquad (Q)\ \sigma_1 \sqsubseteq \sigma_2\ \mathbf{(3)} \qquad Q \sqsubseteq^I Q'\ \mathbf{(4)} \qquad (Q')\ \sigma_1 \mathrel{-\!\!\!\!\sqsubseteq} \sigma_2\ \mathbf{(5)}$$

We first show that $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$ holds. By Property 2.1.3.ii (page 65) applied to (3), we have $\forall\,(Q)\ \sigma_1 \leqslant_/ \forall\,(Q)\ \sigma_2$ **(6)**. By Definition 3.2.1 (page 103), (4), and (1), we get $\forall\,(Q)\ \sigma_2 \sqsubseteq \forall\,(Q')\ \sigma_2$ **(7)**. By Property 2.1.3.ii (page 65) applied to (7), we get $\forall\,(Q)\ \sigma_2 \leqslant_/ \forall\,(Q')\ \sigma_2$ **(8)**. By Property 2.1.3.i (page 65) applied to (5), we get $\forall\,(Q')\ \sigma_1/ = \forall\,(Q')\ \sigma_2/$**(9)**. Let u be an occurrence in $\mathsf{dom}(\forall\,(Q)\ \sigma_1)$ such that $(\forall\,(Q)\ \sigma_1)/u \neq (\forall\,(Q)\ \sigma_2)/u$. By definition of $\leqslant_/$ and (6), we have $(\forall\,(Q)\ \sigma_1)/u = \bot$ and $(\forall\,(Q)\ \sigma_2)/u = g$ **(10)**, where g is not $\bot$. Can u be in $\mathsf{dom}(\sigma_1)$? We would have $\sigma_1/u = \bot$, thus $\forall\,(Q')\ \sigma_1/u = \bot$ **(11)**. However, we have $\forall\,(Q')\ \sigma_2/u = g$ by (10) and (8). This is a contradiction with (11) and (9). Consequently, u is not in $\mathsf{dom}(\sigma_1)$, which means that it is of the form u_1u_2 **(12)** with $\sigma_1/u_1 = \alpha$ **(13)** and $\forall\,(Q)\ \alpha/u_2 = \bot$. By Lemma 3.6.8 (page 115), Q is of the form $(Q_1, \beta \diamond \sigma_b, Q_2)$ **(14)** and u_2 is of the form

$v_1 v_2$ $\mathbf{(\underline{15})}$ such that $\forall\,(Q_2)\ \alpha/v_1 = \beta$ $\mathbf{(\underline{16})}$ and $\sigma_b/v_2 = \bot$ $\mathbf{(\underline{17})}$. Note that σ_b is not in $\mathcal{T}$ by Property 2.1.5.i (page 67). We have $(\forall\,(Q_2)\ \sigma_1/u_1v_1) = \beta$ by (13) and (16). Thus, by Lemma 2.1.4 applied to (3), we get $(\forall\,(Q_2)\ \sigma_2)/u_1v_1 = \beta$ $\mathbf{(\underline{18})}$. Hence, $(\forall\,(Q)\ \sigma_2)/u = \bot$ holds from (12), (15), (18), (14), and (17). This is a contradiction with (10). In summary, we have shown by way of contradiction that $\forall\,(Q)\ \sigma_1/ = \forall\,(Q)\ \sigma_2/$ $\mathbf{(\underline{19})}$ holds. By Lemma 2.7.8, (2), and (5), we get $X \notin w(\sigma_1) - w(\sigma_2)$ $\mathbf{(\underline{20})}$. By Lemma 2.7.8, (2), (3), (19), and (20), we get $(Q)\ \sigma_1 \sqsubseteq \sigma_2$. ■

Proof of Lemma 3.6.10

By hypothesis, $Q_1 \sqsubseteq^I Q_2$ holds. By Lemma 3.6.1 (page 113), there exists a renaming ϕ on $\mathsf{dom}(Q_2)$ and a substitution θ' invariant on I such that $J = \mathsf{dom}(Q_1/I)$ and $Q_1 \sqsubseteq^J \phi(Q_2)\underline{\theta'}$ $\mathbf{(\underline{1})}$ hold. We have $(Q_1)\ \nabla_I \equiv \widehat{Q_1}(\nabla_I)$ by Eq-Mono. By Lemma 3.6.4 and (1), we get $(\phi(Q_2)\underline{\theta'})\ \nabla_I \equiv \widehat{Q_1}(\nabla_I)$. By Property 1.5.11.vii (page 54), we have $\widehat{\phi(Q_2)} \circ \theta'(\nabla_I) = \widehat{\phi(Q_2)} \circ \theta' \circ \widehat{Q_1}(\nabla_I)$ $\mathbf{(\underline{2})}$. Let θ_0 be $\widehat{\phi(Q_2)} \circ \theta'$. Since θ' is invariant on I, (2) gives $\widehat{\phi(Q_2)}(\nabla_I) = \theta_0 \circ \widehat{Q_1}(I)$ $\mathbf{(\underline{3})}$. By Property 3.1.1.iii (page 102), we have $\widehat{\phi(Q_2)} = \phi \circ \widehat{Q_2} \circ \phi^\neg$. Hence, (3) becomes $\phi \circ \widehat{Q_2} \circ \phi^\neg(\nabla_I) = \theta_0 \circ \widehat{Q_1}(I)$ $\mathbf{(\underline{4})}$. Since ϕ is a renaming disjoint from I, we have $\phi^\neg(\nabla_I) = \nabla_I$. Hence (4) gives $\phi \circ \widehat{Q_2}(\nabla_I) = \theta_0 \circ \widehat{Q_1}(I)$. Applying $\phi^\neg$, we get $\widehat{Q_2}(\nabla_I) = \phi^\neg \circ \theta_0 \circ \widehat{Q_1}(\nabla_I)$. Let θ be $\phi^\neg \circ \theta_0$. We get $\widehat{Q_2}(\nabla_I) = \theta \circ \widehat{Q_1}(\nabla_I)$, which implies that $\widehat{Q_2}$ and $\theta \circ \widehat{Q_1}$ are equal on I. ■

Proof of Lemma 3.6.11

First, notice that we can show the lemma for any alpha-conversion of $\forall\,(Q_2)\ \tau_{21} \to \tau_{22}$ (indeed, $\forall\,(Q_2)\ \tau_{21}$ and $\forall\,(Q_2)\ \tau_{22}$ can always be alpha-converted accordingly). Hence, we can freely assume $\forall\,(Q_2)\ \tau_{21} \to \tau_{22}$ to be suitable for Lemma 3.4.4, thus we get a substitution θ such that

$$(Q_2)\ \theta(\tau_{11} \to \tau_{12}) \equiv \tau_{21} \to \tau_{22}\ \ (\underline{\mathbf{1}}) \qquad I = \mathsf{dom}(Q_1/\tau_{11} \to \tau_{12}) \qquad \mathsf{dom}(\theta) \subseteq I$$

$$Q_1 \diamondsuit^{\mathsf{dom}(Q)\cup I}\ Q_2\underline{\theta}\ \ (\underline{\mathbf{2}})$$

By Property 1.5.11.viii applied to (1), we get $(Q_2)\ \theta(\tau_{11}) \equiv \tau_{21}$ $\mathbf{(\underline{3})}$ and $(Q_2)\ \theta(\tau_{12}) \equiv \tau_{22}$. By (2) and Definition 3.2.1, we have $\forall\,(Q_1)\ \tau_{11} \diamondsuit \forall\,(Q_2\underline{\theta})\ \tau_{11}$. By (3), this gives $\forall\,(Q_1)\ \tau_{11} \diamondsuit \forall\,(Q_2)\ \tau_{21}$. We show similarly that $\forall\,(Q_1)\ \tau_{12} \diamondsuit \forall\,(Q_2)\ \tau_{22}$ holds. This is the expected result. ■

Proof of Lemma 3.6.12

We prove the first statement. By hypothesis $Q \sqsubseteq Q_1Q_2$ $\mathbf{(\underline{1})}$, $\mathsf{dom}(Q) = \mathsf{dom}(Q/I)$ $\mathbf{(\underline{2})}$, $I \subseteq \mathsf{dom}(Q_1)$ $\mathbf{(\underline{3})}$, and $\mathsf{dom}(Q_1)\ \#\ \mathsf{dom}(Q_2)$ $\mathbf{(\underline{4})}$ by well-formedness of Q_1Q_2. Thus

$I \# \mathsf{dom}(Q_2)$ $(\underline{\mathbf{5}})$ holds from (3) and (4). Let α' be in $\mathsf{dom}(Q_2/\mathsf{dom}(Q))$ $(\underline{\mathbf{6}})$. By Property 3.3.2.i (page 104), we have $\mathsf{dom}(Q_2/\mathsf{dom}(Q)) = \bigcup_{\alpha\in\mathsf{dom}(Q)} \mathsf{dom}(Q_2/\alpha)$. Hence, from (6), there exists $\alpha \in \mathsf{dom}(Q)$ $(\underline{\mathbf{7}})$ such that we have $\alpha' \in \mathsf{dom}(Q_2/\alpha)$ $(\underline{\mathbf{8}})$. Hence, α is in $\mathsf{dom}(Q) \cap \mathsf{dom}(Q_2)$. Therefore, Q is of the form $(Q_a, \alpha \diamond_1 \sigma_1, Q_b)$ $(\underline{\mathbf{9}})$ and Q_2 of the form $(Q_2^a, \alpha \diamond_2 \sigma_2, Q_2^b)$ $(\underline{\mathbf{10}})$. Additionally, from (8), Q_2^a is of the form $(Q_2^c, \alpha' \diamond_3 \sigma_3, Q_2^d)$ $(\underline{\mathbf{11}})$ and $\alpha' \in \mathsf{ftv}(\forall\,(Q_2^d)\ \sigma_2)$ $(\underline{\mathbf{12}})$. By we have $\alpha \in \mathsf{dom}(Q/I)$ from (2) and (7). By (9), this means that $\alpha \in \mathsf{ftv}(\forall\,(Q_b)\ \nabla_I)$ $(\underline{\mathbf{13}})$. We have the following:

$$\begin{array}{llcll}
(Q) & \forall\,(Q_b)\ \nabla_I & \sqsubseteq & \nabla_I & \text{by \textsc{I-Drop}}^\star \text{ and (9)} \\
(Q_1 Q_2) & \forall\,(Q_b)\ \nabla_I & \sqsubseteq & \nabla_I & \text{by Lemma 3.6.4 and (1)} \\
(Q_1 Q_2^a, \alpha \diamond_2 \sigma_2, Q_2^b) & \forall\,(Q_b)\ \nabla_I & \sqsubseteq & \nabla_I & \text{by (10)} \\
(Q_1 Q_2^a, \alpha \diamond_2 \sigma_2) & \forall\,(Q_2^b Q_b)\ \nabla_I & \sqsubseteq & \forall\,(Q_2^b)\ \nabla_I & \text{by \textsc{R-Context-R}} \\
(Q_1 Q_2^a, \alpha \diamond_2 \sigma_2) & \forall\,(Q_2^b Q_b)\ \nabla_I & \sqsubseteq & \nabla_I & \text{by (5) and \textsc{Eq-Free}}
\end{array}$$

Since we have $\alpha \in \mathsf{dom}(Q_2)$ and (5), we have $\alpha \notin I$, that is, $\alpha \notin \mathsf{ftv}(\nabla_I)$. Hence, by R-Context-R and Eq-Free, we get

$$\begin{array}{llcll}
(Q_1 Q_2^a) & \forall\,(\alpha \diamond_2 \sigma_2)\ \forall\,(Q_2^b Q_b)\ \nabla_I & \sqsubseteq & \nabla_I & \\
(Q_1 Q_2^c, \alpha' \diamond_3 \sigma_3, Q_2^d) & \forall\,(\alpha \diamond_2 \sigma_2)\ \forall\,(Q_2^b Q_b)\ \nabla_I & \sqsubseteq & \nabla_I & \text{by (11)}
\end{array}$$

We get $(Q_1 Q_2^c, \alpha' \diamond_3 \sigma_3)\ \forall\,(Q_2^d)\ \forall\,(\alpha \diamond_2 \sigma_2)\ \forall\,(Q_2^b Q_b)\ \nabla_I \sqsubseteq \forall\,(Q_2^d)\ \nabla_I$ by R-Context-R. By (5) and Eq-Free, we get $(Q_1 Q_2^c, \alpha' \diamond_3 \sigma_3)\ \forall\,(Q_2^d)\ \forall\,(\alpha \diamond_2 \sigma_2)\ \forall\,(Q_2^b Q_b)\ \nabla_I \sqsubseteq \nabla_I$ $(\underline{\mathbf{14}})$. By (13), we have $\alpha \in \mathsf{ftv}(\forall\,(Q_2^b Q_b)\ \nabla_I)$. Hence, by (12), $\alpha' \in \mathsf{ftv}(\forall\,(Q_2^d)\ \forall\,(\alpha \diamond_2 \sigma_2)\ \forall\,(Q_2^b Q_b)\ \nabla_I)$ $(\underline{\mathbf{15}})$. Moreover, $\alpha' \notin I$ by (6) and (5). Hence, $\alpha' \notin \mathsf{ftv}(\nabla_I)$ $(\underline{\mathbf{16}})$. By Lemma 2.1.4 (page 67) on (14), (15), and (16), we get $\sigma_3 \in \mathcal{T}$, which implies $\alpha' \in \mathsf{dom}(\widehat{Q_2})$. This result holds for any α' in $\mathsf{dom}(Q_2/\mathsf{dom}(Q))$, hence we have $\mathsf{dom}(Q_2/\mathsf{dom}(Q)) \subseteq \mathsf{dom}(\widehat{Q_2})$.

We prove the second statement. Let α be in $\mathsf{dom}(Q)$ $(\underline{\mathbf{1}})$. We have to show that $\mathsf{ftv}(\widehat{Q_2}(\alpha)) \subseteq \mathsf{dom}(Q_1)$. It suffices to show that $\mathsf{ftv}(\widehat{Q_2}(\alpha)) \# \mathsf{dom}(Q_2)$ holds. By a way of contradiction, assume that $\beta \in \mathsf{ftv}(\widehat{Q_2}(\alpha))$ $(\underline{\mathbf{2}})$ and $\beta \in \mathsf{dom}(Q_2)$. Then Q_2 is of the form $(Q_a, \beta \diamond \sigma, Q_b)$. By definition of $\widehat{Q_2}$ (which is idempotent) and (2), we must have $\beta \notin \mathsf{dom}(\widehat{Q_2})$ $(\underline{\mathbf{3}})$. From (2), we have $\beta \in \mathsf{ftv}(\widehat{Q_b}(\alpha))$. Hence, $\beta \in \mathsf{ftv}(\forall\,(Q_b)\ \widehat{Q_b}(\alpha))$ (since $\beta \notin \mathsf{dom}(Q_b)$). This gives $\beta \in \mathsf{ftv}(\forall\,(Q_b)\ \alpha)$ $(\underline{\mathbf{4}})$ by Property 1.5.11.vi (page 54) and by observing that $\forall\,(Q_b)\ \alpha \equiv \forall\,(Q_b)\ \widehat{Q_b}(\alpha)$ holds by Eq-Mono. Therefore, from (4), we have $\beta \in \mathsf{dom}(Q_2/\alpha)$, which implies $\beta \in \mathsf{dom}(Q_2/\mathsf{dom}(Q))$ from (1). The first property ensures that $\beta \in \mathsf{dom}(\widehat{Q_2})$, which is a contradiction with (3). We have shown a contradiction, thus the hypothesis $\beta \in \mathsf{ftv}(\widehat{Q_2}(\alpha))$ and $\beta \in \mathsf{dom}(Q_2)$ was wrong. As a consequence, we have $\widehat{Q_2}(\mathsf{dom}(Q)) \subseteq \mathsf{dom}(Q_1)$. ■

Proof of Lemma 3.6.13

By hypothesis, $Q_1 \sqsubseteq^{I \cup J} Q_2Q_3$ holds. By Lemma 3.6.1 (page 113), there exists a renaming ϕ on $\mathsf{dom}(Q_2Q_3)$ and a substitution θ both invariant on $I \cup J$ such that $Q_1 \sqsubseteq^K \phi(Q_2Q_3)\underline{\theta}$ holds, and $K = \mathsf{dom}(Q_1/I \cup J)$ $(\underline{\mathbf{1}})$. Since $\mathsf{ftv}(\tau) \subseteq I \cup J$, we have $\theta(\tau) = \tau$ $(\underline{\mathbf{2}})$. Let Q'_2 be $\phi(Q_2)$ and Q'_3 be $\phi(Q_3)$. We have $Q_1 \sqsubseteq^K Q'_2Q'_3\underline{\theta}$ $(\underline{\mathbf{3}})$. Additionally, (Q_a, Q_b) is $Q_1{\uparrow}I$, thus by Lemma 3.5.2, we have $I \subseteq \mathsf{dom}(Q_a)$ $(\underline{\mathbf{4}})$, $\mathsf{dom}(Q_a/I) = \mathsf{dom}(Q_a)$ $(\underline{\mathbf{5}})$, and $Q_aQ_b \approx Q_1$ $(\underline{\mathbf{6}})$. Hence, (3) becomes $Q_aQ_b \sqsubseteq^K Q'_2Q'_3\underline{\theta}$ $(\underline{\mathbf{7}})$. From (1) and Property 3.3.2.i (page 104), we have $\mathsf{dom}(Q_1/I) \subseteq K$ $(\underline{\mathbf{8}})$. From (5), (4), and (6), we get $\mathsf{dom}(Q_1/I) = \mathsf{dom}(Q_a/I) = \mathsf{dom}(Q_a)$ $(\underline{\mathbf{9}})$. By (9) and (8), we have $\mathsf{dom}(Q_a) \subseteq K$ $(\underline{\mathbf{10}})$. By Property 3.4.2.i (page 106), PE-Free, (7) and (10) we get $Q_a \sqsubseteq Q'_2Q'_3\underline{\theta}$ $(\underline{\mathbf{11}})$. Hence, we get $(Q_a, \gamma \geq \forall\,(Q_b)\ \tau) \sqsubseteq (Q'_2Q'_3\underline{\theta}, \gamma \geq \forall\,(Q_b)\ \tau)$ by Property 3.4.2.iii (page 106). Let θ' be $\widehat{Q'_3} \circ \theta$. By Eq-Mono*, we get $(Q_a, \gamma \geq \forall\,(Q_b)\ \tau) \sqsubseteq (Q'_2Q'_3\underline{\theta}, \gamma \geq \theta'(\forall\,(Q_b)\ \tau))$ $(\underline{\mathbf{12}})$. We also have

$$\begin{array}{llcll}
(Q_aQ_b) & \forall\,(Q_b)\ \tau & \sqsubseteq & \tau & \text{by I-Drop}^\star \\
(Q'_2Q'_3\underline{\theta}) & \forall\,(Q_b)\ \tau & \sqsubseteq & \tau & \text{by Lemma 3.6.4 and (7)} \\
(Q'_2) & \forall\,(Q'_3)\ \theta(\forall\,(Q_b)\ \tau) & \sqsubseteq & \forall\,(Q'_3)\ \theta(\tau) & \text{by R-Context-R} \\
(Q'_2) & \forall\,(Q'_3)\ \theta'(\forall\,(Q_b)\ \tau) & \sqsubseteq & \forall\,(Q'_3)\ \theta'(\tau) & \text{by Eq-Mono}^\star
\end{array}$$

Hence, $(Q'_2)\ \forall\,(Q'_3)\ \theta'(\forall\,(Q_b)\ \tau) \sqsubseteq \forall\,(Q'_3)\ \theta'(\tau)$ $(\underline{\mathbf{13}})$ holds. We have $I \subseteq \mathsf{dom}(Q_2)$ by hypothesis, and ϕ disjoint from I, thus $I \subseteq \mathsf{dom}(Q'_2)$ $(\underline{\mathbf{14}})$. By Lemma 3.6.12 (page 116) applied to (11), (14) and (5), we get $\mathsf{dom}(Q'_3\underline{\theta}/\mathsf{dom}(Q_a)) \subseteq \mathsf{dom}(\theta')$ and $\theta'(\mathsf{dom}(Q_a)) \subseteq \mathsf{dom}(Q'_2)$ $(\underline{\mathbf{15}})$. Consider the instantiation (13). The free variables of $\forall\,(Q_b)\ \tau$ are in $\mathsf{dom}(Q_a)$ (since $Q_aQ_b \approx Q_1$). Hence, the free variables of $\theta'(\forall\,(Q_b)\ \tau)$ are in $\theta'(\mathsf{dom}(Q_a))$. By (15), they are in $\mathsf{dom}(Q'_2)$. In particular, they are not in $\mathsf{dom}(Q'_3)$ (since $Q'_2 \# Q'_3$ holds by well-formedness of (3)). Hence, $\forall\,(Q'_3)\ \theta'(\forall\,(Q_b)\ \tau) \equiv \theta'(\forall\,(Q_b)\ \tau)$ holds by Eq-Free. Consequently, (13) gives

$$(Q'_2)\ \theta'(\forall\,(Q_b)\ \tau) \sqsubseteq \forall\,(Q'_3)\ \theta'(\tau) \qquad (\underline{\mathbf{16}})$$

Additionally, we have

$$\begin{array}{lcll}
(Q_a, \gamma \geq \forall\,(Q_b)\ \tau) & \sqsubseteq & (Q'_2, \gamma \geq \theta'(\forall\,(Q_b)\ \tau), Q'_3\underline{\theta}) & \text{by PE-Comm on (12).} \\
(Q_a, \gamma \geq \forall\,(Q_b)\ \tau) & \sqsubseteq & (Q'_2, \gamma \geq \forall\,(Q'_3)\ \theta'(\tau), Q'_3\underline{\theta}) & \text{by PI-Context-L} \\
 & & & \text{and (16).} \\
(Q_a, \gamma \geq \forall\,(Q_b)\ \tau) & \sqsubseteq & (Q'_2, \gamma \geq \forall\,(Q'_3)\ \theta(\tau), Q'_3\underline{\theta}) & \text{by Eq-Mono}^\star. \\
(Q_a, \gamma \geq \forall\,(Q_b)\ \tau) & \sqsubseteq & (Q'_2, \gamma \geq \forall\,(Q'_3)\ \tau, Q'_3\underline{\theta}) & \text{by (2).} \\
(Q_a, \gamma \geq \forall\,(Q_b)\ \tau) & \sqsubseteq & (\phi(Q_2), \gamma \geq \forall\,(\phi(Q_3))\ \tau, Q'_3\underline{\theta}) & \text{by definition.} \\
(Q_a, \gamma \geq \forall\,(Q_b)\ \tau) & \sqsubseteq & (\phi(Q_2), \gamma \geq \phi(\forall\,(Q_3)\ \tau), Q'_3\underline{\theta}) & \text{by alpha-renaming.} \\
(Q_a, \gamma \geq \forall\,(Q_b)\ \tau) & \sqsubseteq & (\phi(Q_2, \gamma \geq \forall\,(Q_3)\ \tau), Q'_3\underline{\theta}) &
\end{array}$$

Then $(Q_a, \gamma \geq \forall\,(Q_b)\ \tau) \sqsubseteq^{I \cup \{\gamma\}} (\phi(Q_2, \gamma \geq \forall\,(Q_3)\ \tau))$ $(\underline{\mathbf{17}})$ holds by PE-Free and Property 3.4.2.i (page 106). By Property 3.4.2.iv (page 106), we have $(Q_2, \gamma \geq \forall\,(Q_3)\ \tau) \equiv (\phi(Q_2, \gamma \geq \forall\,(Q_3)\ \tau)\underline{\phi})$, thus $(Q_2, \gamma \geq \forall\,(Q_3)\ \tau) \equiv^{I \cup \{\gamma\}} (\phi(Q_2, \gamma \geq \forall\,(Q_3)\ \tau))$ holds by PE-Free and Property 3.4.2.i (page 106). With (17), we get $(Q_a, \gamma \geq \forall\,(Q_b)\ \tau) \sqsubseteq^{I \cup \{\gamma\}} (Q_2, \gamma \geq \forall\,(Q_3)\ \tau)$. This is the expected result. ■

Proof of Lemma 4.2.4

By Lemma 3.6.1 (page 113), there exists a renaming ϕ on $\mathsf{dom}(Q_2)$ and a substitution θ invariant on I such that $Q_1 \sqsubseteq^J \phi(Q_2)\underline{\theta}$ $(\underline{\mathbf{1}})$ holds, and J is $\mathsf{dom}(Q_1/I)$. Let (Q_3, Q_4) be $Q_1 \uparrow \mathsf{ftv}(\sigma')$. By Lemma 3.5.2, we have $Q_3Q_4 \approx Q_1$ $(\underline{\mathbf{2}})$ and $\mathsf{ftv}(\sigma') \# \mathsf{dom}(Q_4)$ $(\underline{\mathbf{3}})$. Then Q_4 is $(Q_5, \alpha \diamond \sigma, Q_6)$ and Q_1 is a rearrangement of $(Q_3Q_5, \alpha \diamond \sigma, Q_6)$ by (2). From (3), we have $\mathsf{ftv}(\sigma') \# \mathsf{dom}(Q_6)$ $(\underline{\mathbf{4}})$. Let Q_a be Q_3Q_5 and Q_b be Q_6 $(\underline{\mathbf{5}})$. Then Q_1 is a rearrangement of $(Q_a, \alpha \diamond \sigma, Q_b)$ $(\underline{\mathbf{6}})$. By hypothesis, if $\diamond$ is $=$, then $(Q_1)\ \sigma \mathrel{\sqsubset\!\!\!\!-} \sigma'$ holds. This implies that $(Q_1)\ \sigma \mathrel{\sqsubset\!\!\!\!-}^{?} \sigma'$ returns true, by Lemma 4.2.2. Hence, $Q_1 \Leftarrow (\alpha \diamond' \sigma')$ is well-defined (that is, it does not fail on the fourth step). We write it Q_1'.

By definition, Q_1' is $(Q_a, \alpha \diamond' \sigma', Q_b)$ $(\underline{\mathbf{7}})$ and $\mathsf{dom}(Q_b) \# \mathsf{ftv}(\sigma')$ holds from (4) and (5).

We have the following:

$$\begin{array}{rrcrl}
(Q_a, \alpha \diamond \sigma, Q_b) & \forall\,(Q_b)\ \nabla_I & \sqsubseteq & \nabla_I & \text{by I-Drop}^\star. \\
(Q_1) & \forall\,(Q_b)\ \nabla_I & \sqsubseteq & \nabla_I & \text{by (6)} \\
 & & & & \text{and Property 1.7.2.ii} \\
(\phi(Q_2)\underline{\theta}) & \forall\,(Q_b)\ \nabla_I & \sqsubseteq & \nabla_I & \text{by Lemma 3.6.4 on (1).} \\
 & \forall\,(\phi(Q_2)\underline{\theta}Q_b)\ \nabla_I & \sqsubseteq & \forall\,(\phi(Q_2)\underline{\theta})\ \nabla_I & \text{by R-Context-R} \\
 & \forall\,(\phi(Q_2)\underline{\theta}Q_b)\ \nabla_I & \sqsubseteq & \forall\,(\phi(Q_2))\ \nabla_I & \text{since } \mathsf{dom}(\theta) \# I. \\
 & \forall\,(\phi(Q_2)\underline{\theta}Q_b)\ \nabla_I & \sqsubseteq & \forall\,(Q_2)\ \nabla_I & \text{by alpha-conversion}
\end{array}$$

Hence, $\forall\,(\phi(Q_2)\underline{\theta}Q_b)\ \nabla_I \sqsubseteq \forall\,(Q_2)\ \nabla_I$ $(\underline{\mathbf{8}})$ holds. By hypothesis, we have $(Q_2)\ \sigma' \sqsubseteq \alpha$ $(\underline{\mathbf{9}})$ and $(Q_2)\ \sigma' \mathrel{\sqsubset\!\!\!\!-} \alpha$ $(\underline{\mathbf{10}})$ when $\diamond'$ is rigid. We have $Q_2 \equiv^I \phi(Q_2)\underline{\phi}$ by Property 3.4.2.iv (page 106). We get $Q_2 \equiv^I \phi(Q_2)\underline{\theta}$ $(\underline{\mathbf{11}})$ by PE-Free (twice). Hence, we have $(\phi(Q_2)\underline{\theta})\ \sigma' \sqsubseteq \alpha$ $(\underline{\mathbf{12}})$ (and possibly $(\phi(Q_2)\underline{\theta})\ \sigma' \mathrel{\sqsubset\!\!\!\!-} \alpha$ $(\underline{\mathbf{13}})$) by Lemma 3.6.4, (9), possibly (10), and (11). Then we have the following (we abbreviated R-Context-Flexible and R-Context-Rigid into RC-Flexible and RC-Rigid, respectively):

$$\begin{array}{rrcrl}
(Q_a, \alpha \diamond \sigma, Q_b) & \forall\,(Q_1')\ \nabla_I & \sqsubseteq & \forall\,(\alpha \diamond' \sigma', Q_b)\ \nabla_I & \text{by I-Drop}^\star \text{ and (7)} \\
(Q_1) & \forall\,(Q_1')\ \nabla_I & \sqsubseteq & \forall\,(\alpha \diamond' \sigma', Q_b)\ \nabla_I & \text{by (6)} \\
(\phi(Q_2)\underline{\theta}) & \forall\,(Q_1')\ \nabla_I & \sqsubseteq & \forall\,(\alpha \diamond' \sigma', Q_b)\ \nabla_I & \text{by Lemma 3.6.4 and (1)} \\
(\phi(Q_2)\underline{\theta}) & \forall\,(Q_1')\ \nabla_I & \sqsubseteq & \forall\,(\alpha \diamond' \alpha, Q_b)\ \nabla_I & \text{by RC-Flexible and (12)} \\
 & & & & \text{or RC-Rigid and (13)} \\
(\phi(Q_2)\underline{\theta}) & \forall\,(Q_1')\ \nabla_I & \sqsubseteq & \forall\,(Q_b)\ \nabla_I & \text{by Eq-Mono}^\star \\
 & \forall\,(Q_1')\ \nabla_I & \sqsubseteq & \forall\,(\phi(Q_2)\underline{\theta}Q_b)\ \nabla_I & \text{by R-Context-R}
\end{array}$$

The left-hand side $\forall\,(Q'_1)\ \nabla_I$ is closed, thus $\forall\,(\phi(Q_2)\underline{\theta})\ \forall\,(Q'_1)\ \nabla_I$ and $\forall\,(Q'_1)\ \nabla_I$ are equivalent by Eq-Free. This is why we get the last line by Eq-Free. Finally, combining with (8), we get $\forall\,(Q'_1)\ \nabla_I \sqsubseteq \forall\,(Q_2)\ \nabla_I$, thus Property 3.4.5.i (page 109) gives the expected result. ■

Proof of Lemma 4.2.5

By hypothesis, we have $(Q)\ \sigma \sqsubseteq \sigma'$ **($\underline{1}$)**. Let (Q_1, Q_2) be $Q\!\uparrow \mathsf{ftv}(\sigma')$. By Lemma 3.5.2, we have $Q_1 Q_2 \approx Q$. By hypothesis, we have Q_2 of the form $(Q_2^a, \alpha \diamond \sigma, Q_2^b)$. If $\diamond$ is $=$, then $(Q)\ \sigma \mathrel{\sqsubseteq\!\!\!\!\!-}^{?} \sigma'$ returns true, thus we have $(Q)\ \sigma \mathrel{\sqsubseteq\!\!\!\!\!-} \sigma'$ **($\underline{2}$)** by Lemma 4.2.2 and (1). Finally, the algorithm returns $Q' = (Q_1 Q_2^a, \alpha \diamond \sigma', Q_2^b)$. The prefix $(Q_1 Q_2^a, \alpha \diamond \sigma, Q_2^b)$ is a rearrangement of Q. Besides, $(Q_1 Q_2^a, \alpha \diamond \sigma, Q_2^b) \sqsubseteq Q'$ holds by PA-Context-L and (2) or PI-Context-L and (1). Hence, $Q \sqsubseteq Q'$ holds, which is the expected result. ■

Proof of Lemma 4.2.7

By hypothesis, Q is of the form $(Q_0, \alpha \diamond \sigma, Q_1, \alpha' \diamond' \sigma, Q_2)$, or $(Q_0, \alpha' \diamond' \sigma, Q_1, \alpha \diamond \sigma, Q_2)$. By well-formedness of Q, we must have $\mathsf{ftv}(\sigma)\ \#\ \mathsf{dom}(Q_1)$. Hence, by Eq-Comm, we get $Q \equiv (Q_0, \alpha \diamond \sigma, \alpha' \diamond' \sigma, Q_1 Q_2)$ **($\underline{1}$)**. If $\diamond$ and $\diamond'$ are flexible, then $\diamond''$ is $\geq$ and $Q \sqsubseteq (Q_0, \alpha \diamond'' \sigma, \alpha' \diamond'' \sigma, Q_1 Q_2)$ **($\underline{2}$)** holds from (1) by PE-Refl. Otherwise, $\diamond''$ is $=$ and (2) holds from (1) and by Rule PI-Rigid (if necessary). Finally, $Q \sqsubseteq (Q_0, \alpha \diamond'' \sigma, \alpha' = \alpha, Q_1 Q_2)$ **($\underline{3}$)** holds from (2), by PI-Trans, PI-Context-L on the binding $(\alpha' \geq \sigma)$, and I-Hyp when $\diamond''$ is flexible, or PA-Context-L on the binding $(\alpha' = \sigma)$ and A-Hyp when $\diamond''$ is rigid. Hence, $Q \sqsubseteq Q'$ holds from (3). Additionally, $(Q')\ \alpha \equiv \alpha'$ holds by Eq-Mono. ■

Proof of Lemma 4.2.8

Merging is only a particular case of *Updating*. More precisely, let $\diamond''$ be $\geq$ if both $\diamond$ and $\diamond'$ are $\geq$, and $=$ otherwise. Let Q_a be $Q_1 \Leftarrow (\alpha \diamond'' \sigma)$, let Q_b be $Q_a \Leftarrow (\alpha \diamond'' \sigma)$, and Q_c be $Q_b \Leftarrow (\alpha' = \alpha)$. We show that Q_a, Q_b, and Q_c are well-defined (that is, the updates do not fail) and that Q_c is a rearrangement of $Q \Leftarrow \alpha \wedge \alpha'$. By hypothesis and without loss of generality Q_1 is of the form $(Q_4, \alpha \diamond \sigma, Q_5, \alpha' \diamond' \sigma, Q_6)$ **($\underline{1}$)**. By well-formedness, we have $\alpha' \notin \mathsf{dom}(Q_1/\sigma)$, thus $\alpha' \notin \mathsf{dom}(Q_b/\alpha)$. This implies that $Q_b \Leftarrow (\alpha' = \alpha)$ can be applied. Let (P_1, P_2) be $Q_1\!\uparrow \mathsf{ftv}(\sigma)$. By Lemma 3.5.2, we have $P_1 P_2 \approx Q_1$ and $\mathsf{dom}(P_1) = \mathsf{dom}(Q_1/\sigma)$ **($\underline{2}$)**. Besides, by well-formedness of (1), we must have $\mathsf{ftv}(\sigma) \subseteq \mathsf{dom}(Q_4)$ **($\underline{3}$)**. More precisely, we must have $\mathsf{dom}(Q_1/\sigma) \subseteq \mathsf{dom}(Q_4)$ **($\underline{4}$)**. Hence $\mathsf{dom}(P_1) \subseteq \mathsf{dom}(Q_4)$ **($\underline{5}$)** holds from (4) and (2). By Lemma 4.2.2, $(Q)\ \sigma \mathrel{\sqsubseteq\!\!\!\!\!-}^{?} \sigma$ holds. Hence, the algorithm does not fail on its fourth step. Finally, it returns Q_a, which is a rearrangement of $(Q_4, \alpha \diamond'' \sigma, Q_5, \alpha' \diamond' \sigma, Q_6)$. Similarly, Q_b is a rearrangement of $(Q_4, \alpha \diamond'' \sigma, Q_5, \alpha' \diamond'' \sigma, Q_6)$ $\mathbf{(\underline{6})}$. Since $\mathsf{ftv}(\sigma) \subseteq \mathsf{dom}(Q_4)$ (from (3)), Q_b is also a rearrangement of $(Q_4, \alpha\diamond''\sigma, \alpha'\diamond''\sigma, Q_5Q_6)$. Then observing that $(Q_b)\ \sigma \mathrel{-\!\!\!\!\sqsubset} \alpha$ holds when $\diamond''$ is rigid, we see that $Q_b \Leftarrow (\alpha' = \alpha)$ is well-defined, that is, Q_c is well-defined, and is a rearrangement of $(Q_4, \alpha\diamond''\sigma, \alpha' = \alpha, Q_4Q_6)$, that is, a rearrangement of $Q \Leftarrow \alpha \wedge \alpha'$ $\mathbf{(\underline{7})}$. By Lemma 3.6.1 (page 113), there exists a renaming ϕ on $\mathsf{dom}(Q_2)$ and a substitution θ invariant on I such that $Q_1 \sqsubseteq^J \phi(Q_2)\underline{\theta}$ $\mathbf{(\underline{8})}$ holds, and J is $\mathsf{dom}(Q_1/I)$. Hence, $\mathsf{ftv}(\sigma) \cup \{\alpha\} \subseteq J$ $\mathbf{(\underline{9})}$ holds from the hypothesis $\alpha, \alpha' \in I$. We have $(Q_1)\ \sigma \sqsubseteq \alpha$ by I-Hyp, thus $(\phi(Q_2)\underline{\theta})\ \sigma \sqsubseteq \alpha$ holds by Lemma 3.6.4 and (8). We have $(Q_2)\ \alpha \equiv \alpha'$ by hypothesis, and $Q_2 \equiv^I \phi(Q_2)\underline{\theta}$ $\mathbf{(\underline{10})}$ is derivable by Property 3.4.2.iv (page 106) and PE-Free (observing that ϕ and θ are invariant on I), thus $(\phi(Q_2)\underline{\theta})\ \alpha \equiv \alpha'$ $\mathbf{(\underline{11})}$ holds. Besides, if $\diamond''$ is $=$, then either $\diamond$ or $\diamond'$ is $=$. In the first case, we have $(Q_1)\ \sigma \mathrel{-\!\!\!\!\sqsubset} \alpha$ by A-Hyp. In the second case, we have $(Q_1)\ \sigma \mathrel{-\!\!\!\!\sqsubset} \alpha'$ by A-Hyp. Then $(\phi(Q_2)\underline{\theta})\ \sigma \mathrel{-\!\!\!\!\sqsubset} \alpha$ or $(\phi(Q_2)\underline{\theta})\ \sigma \mathrel{-\!\!\!\!\sqsubset} \alpha'$ $\mathbf{(\underline{12})}$ holds by Lemma 3.6.4 and (8). By (12) and (11), we get $(\phi(Q_2)\underline{\theta})\ \sigma \mathrel{-\!\!\!\!\sqsubset} \alpha$ $\mathbf{(\underline{13})}$. Note that $(Q_1)\ \sigma \mathrel{-\!\!\!\!\sqsubset} \sigma$ $\mathbf{(\underline{14})}$ and $(Q_1)\ \sigma \sqsubseteq \sigma$ $\mathbf{(\underline{15})}$ hold by Eq-Refl, A-Equiv, and I-Equiv*. By Lemma 4.2.4 (page 123), (8), (1), (14), (15), (9), and (13) we have $Q_a \sqsubseteq^J \phi(Q_2)\underline{\theta}$. Similarly, $Q_b \sqsubseteq^J \phi(Q_2)\underline{\theta}$ is derivable. By Property 3.4.2.i (page 106) and (10), we get $Q_b \sqsubseteq^I Q_2$ $\mathbf{(\underline{16})}$. From (6), $(Q_b)\ \sigma \sqsubseteq \alpha'$ $\mathbf{(\underline{17})}$ holds by I-Hyp when $\diamond''$ is flexible, and $(Q_b)\ \sigma \mathrel{-\!\!\!\!\sqsubset} \alpha'$ $\mathbf{(\underline{18})}$ holds by A-Hyp when $\diamond''$ is rigid. By well-formedness of Q_b, we have $\alpha' \notin \mathsf{dom}(Q_b/\alpha)$ $\mathbf{(\underline{19})}$. We have by hypothesis $\alpha, \alpha' \in I$ $\mathbf{(\underline{20})}$ and $(Q_2)\ \alpha \equiv \alpha'$ $\mathbf{(\underline{21})}$. By Lemma 4.2.4 (page 123), (16), (6), (17), (18), (19), (20), (21), we get $Q_c \sqsubseteq^I Q_2$. Hence, $(Q_1 \Leftarrow \alpha \wedge \alpha') \sqsubseteq^I Q_2$ holds from (7). ■

Proof of Property 4.3.1

We prove both properties simultaneously, by induction on the recursive calls to `unify` and `polyunify`. ■

Proof of Property 4.3.2

By induction on the recursive calls to `unify`. All cases are easy. ■

Proof of Lemma 4.5.1

We prove the result for `unify`′ and `polyunify`′ simultaneously, by induction on the recursive calls. We get the result for `polyunify`′ by induction hypothesis and Property 4.3.2.i (page 126). As for `unify`′, we proceed by case analysis on (τ_1, τ_2). Cases (α, α) and $(g_1\ \ldots, g_2\ \ldots)$ with $g_1 \neq g_2$ are immediate (`unify` and `unify`′ return the same result). Cases $(g\ \tau_1^1\ \ldots, g\ \tau_2^1\ \ldots)$ and (α, τ) or (τ, α) with $(\alpha \diamond \sigma) \in Q$ and $\sigma \in \mathcal{V}$ are by induction hypothesis (`unify` and `unify`′ are identical).

∘ Case (α, τ) or (τ, α) with $(\alpha \diamond \tau') \in Q$: In this case, `unify` simply returns `unify` (Q, τ, τ'), while `unify`$'$ is more elaborate. We have to consider the two following subcases:

Subcase $\alpha \in \mathsf{dom}(Q/\tau)$ $\mathbf{(\underline{1})}$: By Property 1.5.11.vii (page 54), we have $\widehat{Q}(\tau) = \widehat{Q}(\alpha)$ $\mathbf{(\underline{2})}$ if and only if $(Q)\ \tau \equiv \alpha$ holds. If (2) does not hold, then `unify`$'$ fails. We have to show that `unify` fails too. By a way of contradiction, assume `unify` succeeds with a prefix Q'. Then by soundness (Lemma 4.4.1), we would have $Q \sqsubseteq Q'$ $\mathbf{(\underline{3})}$ and $(Q')\ \alpha \equiv \tau$ $\mathbf{(\underline{4})}$. From (1), Q is of the form $(Q_a, \alpha \diamond \sigma, Q_b)$ and $\alpha \in \mathsf{ftv}(\forall\,(Q_b)\ \tau)$ $\mathbf{(\underline{5})}$. Hence, we can derive $(Q)\ \forall\,(Q_b)\ \tau \sqsubseteq \tau$ $\mathbf{(\underline{6})}$ by I-Drop*. By Lemma 3.6.4, (6), and (3), we get $(Q')\ \forall\,(Q_b)\ \tau \sqsubseteq \tau$. From (4), we get $(Q')\ \forall\,(Q_b)\ \tau \sqsubseteq \alpha$ $\mathbf{(\underline{7})}$ By Property 2.1.7.ii (page 68), (7), and (5), we get $\forall\,(Q_b)\ \tau \equiv \alpha$. Hence, (6) gives $(Q)\ \alpha \sqsubseteq \tau$. By Lemma 2.1.6, it leads to $(Q)\ \alpha \equiv \tau$. This is a contradiction with the hypothesis that (2) does not hold. Hence, `unify` cannot succeed, and this is the expected result.

Otherwise, (2) holds, and `unify`$'$ returns Q. From (2), we have $(Q)\ \alpha \equiv \tau$ $\mathbf{(\underline{8})}$. We have to show that `unify` returns a rearrangement of Q. From (1), Q is of the form $(Q_a, \alpha \diamond \sigma, Q_b)$ and $\alpha \in \mathsf{ftv}(\forall\,(Q_b)\ \tau)$ $\mathbf{(\underline{9})}$. Hence, we can derive $(Q)\ \forall\,(Q_b)\ \tau \sqsubseteq \tau$ by I-Drop*. From (8), we get $(Q)\ \forall\,(Q_b)\ \tau \sqsubseteq \alpha$ $\mathbf{(\underline{10})}$ By Property 2.1.7.ii (page 68), (10), and (9), we get $\forall\,(Q_b)\ \tau \equiv \alpha$ $\mathbf{(\underline{11})}$. This implies that either τ is α, or it is a variable β such that $\beta \in \mathsf{dom}(Q_b)$. In the first case, `unify` (Q, τ, α) returns Q, which is the expected result. In the second case, Q_b is of the form $(Q_b^1, \beta \diamond_b \sigma_b, Q_b^2)$. From (11) and Eq-Var, we get $\forall\,(Q_b^1)\ \sigma_b \equiv \alpha$. This implies $\sigma_b \in \mathcal{V}$, that is, $\sigma_b \equiv \gamma$ such that $\gamma \in \mathsf{dom}(Q_b^1)$.

Subcase $\alpha \notin \mathsf{dom}(Q/\tau)$:

∘ Case (α, τ) or (τ, α) with $(\alpha \diamond \sigma) \in Q$ $\mathbf{(\underline{12})}$: Both algorithms fail in the first step, or both continue. Let I be $\{\beta \in \mathsf{dom}(Q) \mid \beta \neq \alpha \text{ and } \alpha \notin \mathsf{dom}(Q/\beta)\}$ and (Q_1, Q_2) be $Q{\uparrow}I$. The algorithm `unify`$'$ calls `polyunify`$'$ (Q_1, σ, τ) while `unify` calls `polyunify` (Q, σ, τ). By Lemma 3.5.2, Q_1Q_2 is a rearrangement of Q, $I \subseteq \mathsf{dom}(Q_1)$ and $\mathsf{dom}(Q_1/I) = \mathsf{dom}(Q_1)$. We have $\alpha \notin \mathsf{dom}(Q/\sigma)$ by well-formedness of Q and (12), hence $\mathsf{ftv}(\sigma) \subseteq I$. We have $\alpha \notin \mathsf{dom}(Q/\tau)$ (otherwise both algorithms fail in the first step), hence $\mathsf{ftv}(\tau) \subseteq I$. Thus, we have $\mathsf{ftv}(\tau) \cup \mathsf{ftv}(\sigma) \subseteq \mathsf{dom}(Q_1)$. By Property 4.3.2.ii (page 126) and induction hypothesis, `polyunify`$'$ (Q_1, σ, τ) returns Q_1' if and only if `polyunify` (Q, σ, τ) returns Q', a rearrangement of $Q_1'Q_2$. Then $Q_1'Q_2 \Leftarrow (\alpha = \tau)$ is a rearrangement of $Q' \Leftarrow (\alpha = \tau)$ by Lemma 4.2.3 (page 123). This is the expected result.

∘ Case (α_1, α_2): Both algorithms fail if $\alpha_1 \in \mathsf{dom}(Q/\sigma_2)$ or $\alpha_2 \in \mathsf{dom}(Q/\sigma_1)$. Let I be $\{\beta \in \mathsf{dom}(Q) \mid \beta \notin \{\alpha_1, \alpha_2\} \text{ and } \{\alpha_1, \alpha_2\} \# \mathsf{dom}(Q/\beta)\}$. Let (Q_1, Q_2) be $Q{\uparrow}I$. The algorithm `unify`$'$ calls `polyunify`$'$ $(Q_1, \sigma_1, \sigma_2)$, while `unify` calls `polyunify` (Q, σ_1, σ_2). By Lemma 3.5.2 and by definition of I, Q_1Q_2 is a rearrangement of Q, and we have $\mathsf{ftv}(\sigma_1) \cup \mathsf{ftv}(\sigma_2) \subseteq \mathsf{dom}(Q_1)$. Hence, by Property 4.3.2.i (page 126) and induction hypothesis, `polyunify`$'$ $(Q_1, \sigma_1, \sigma_2)$ returns (Q_1', σ_3') if and only if `polyunify` (Q, σ_1, σ_2) returns (Q', σ_3) such that Q' is a rearrangement of $Q_1'Q_2$ and σ_3 is a rearrangement of σ'_3. Hence, $Q'_1Q_2 \Leftarrow (\alpha_1 \diamond_1 \sigma'_3) \Leftarrow (\alpha_2 \diamond_2 \sigma'_3) \Leftarrow \alpha_1 \wedge \alpha_2$ is a rearrangement of $Q' \Leftarrow (\alpha_1 \diamond_1 \sigma_3) \Leftarrow (\alpha_2 \diamond_2 \sigma_3) \Leftarrow \alpha_1 \wedge \alpha_2$. This is the expected result. ■

Proof of Property 4.5.3

Properties i, ii, and iii are immediate.

We show Properties iv and v simultaneously, by induction on the recursive calls to `unify′` and `polyunify′`. For `unify′`, we proceed by case on (τ_1, τ_2):

- Case (α, α): We return Q and we have $\lceil Q \rceil \leqslant \lceil Q \rceil$.
- Case $(g\ \tau_1^1\ ..\ \tau_1^n, g\ \tau_2^1\ ..\ \tau_2^n)$: by induction hypothesis.
- Case $(g_1\ \tau_1^1\ ..\ \tau_1^p, g_2\ \tau_2^1\ ..\ \tau_2^q)$ with $g_1 \neq g_2$: is not possible since, by hypothesis, we succeed.
- Case (α, τ) or (τ, α) with $(\alpha \diamond \sigma) \in Q$ and $\sigma \in \mathcal{V}$: By induction hypothesis.
- Case (α, τ) or (τ, α): We have $(Q_1, Q_2) = Q{\uparrow}I$. By Lemmas 3.5.2 and ii, we get $\lceil Q \rceil = \lceil Q_1 \rceil + \lceil Q_2 \rceil$ $\mathbf{(\underline{1})}$. Note that $(\alpha \diamond \sigma) \in Q_2$. We have $(Q'_1, \sigma') = $ `polyunify′` (Q_1, σ, τ). By induction hypothesis, $\lceil Q'_1 \rceil + \lceil \sigma' \rceil \leqslant \lceil Q_1 \rceil + \lceil \sigma \rceil + \lceil \tau \rceil$, that is, $\lceil Q'_1 \rceil + \lceil \sigma' \rceil \leqslant \lceil Q_1 \rceil + \lceil \sigma \rceil$. Hence, we have $\lceil Q'_1 \rceil \leqslant \lceil Q_1 \rceil + \lceil \sigma \rceil$ $\mathbf{(\underline{2})}$. Let Q' be $Q'_1Q_2 \Leftarrow (\alpha = \tau)$. We note that $\lceil Q' \rceil = \lceil Q'_1 \rceil + \lceil Q_2 \rceil - \lceil \sigma \rceil + \lceil \tau \rceil$. Hence, by (2), we get $\lceil Q' \rceil \leqslant \lceil Q_1 \rceil + \lceil \sigma \rceil + \lceil Q_2 \rceil - \lceil \sigma \rceil$, that is, $\lceil Q' \rceil \leqslant \lceil Q_1 \rceil + \lceil Q_2 \rceil$. By (1), this gives $\lceil Q' \rceil \leqslant \lceil Q \rceil$.
- Case (α_1, α_2): Similarly, $\lceil Q \rceil = \lceil Q_1 \rceil + \lceil Q_2 \rceil$ and by induction hypothesis, $\lceil Q'_1 \rceil + \lceil \sigma_3 \rceil \leqslant \lceil Q_1 \rceil + \lceil \sigma_1 \rceil + \lceil \sigma_2 \rceil$ $\mathbf{(\underline{3})}$. Let Q' be $Q'_1Q_2 \Leftarrow (\alpha_1 \diamond_1 \sigma_3) \Leftarrow (\alpha_2 \diamond_2 \sigma_3) \Leftarrow \alpha_1 \wedge \alpha_2$. We note that $\lceil Q' \rceil = \lceil Q'_1 \rceil + \lceil Q_2 \rceil - \lceil \sigma_1 \rceil - \lceil \sigma_2 \rceil + \lceil \sigma_3 \rceil$. Hence, $\lceil Q' \rceil \leqslant \lceil Q_1 \rceil + \lceil Q_2 \rceil$ holds from (3), that is, $\lceil Q' \rceil \leqslant \lceil Q \rceil$. This is the expected result.

For `polyunify`, we have $\sigma_1 = \forall\,(Q_1)\ \tau_1$ and $\sigma_2 = \forall\,(Q_2)\ \tau_2$. We compute $Q_0 = $ `unify` $(QQ_1Q_2, \tau_1, \tau_2)$ and $(Q_3, Q') = Q_0{\uparrow}\,\mathsf{dom}(Q)$. By Lemma 3.5.2, $Q_3Q' \approx Q_0$. Hence, by Property ii, we get $\lceil Q_0 \rceil = \lceil Q_3 \rceil + \lceil Q' \rceil$ $\mathbf{(\underline{1})}$. We have $\sigma_3 = \forall\,(Q')\ \tau_1$, thus $\lceil \sigma_3 \rceil = \lceil Q' \rceil$ $\mathbf{(\underline{2})}$ by Property i. By induction hypothesis, we have $\lceil Q_0 \rceil \leqslant \lceil QQ_1Q_2 \rceil$ $\mathbf{(\underline{3})}$. By (3), (1), (2), we get $\lceil Q_3 \rceil + \lceil \sigma_3 \rceil \leqslant \lceil QQ_1Q_2 \rceil$. By Properties ii and i, we get $\lceil Q_3 \rceil + \lceil \sigma_3 \rceil \leqslant \lceil Q \rceil + \lceil \sigma_1 \rceil + \lceil \sigma_2 \rceil$. This is the expected result.

■

Proof of Lemma 4.6.1

In the cases $(\bot, \sigma)$, or $(\sigma, \bot)$, `polyunify` returns σ, which is σ_1 or σ_2, thus it is not in $\mathcal{V}$ by hypothesis. In the last case, it returns $\forall\,(Q_4)\ \tau_1$, where the constructed form of σ_1 is $\forall\,(Q_1)\ \tau_1$. Since σ_1 is not in $\mathcal{V}$, we must have $\tau_1 \notin \vartheta$, thus $\forall\,(Q_4)\ \tau_1 \notin \mathcal{V}$. ■

Proof of Corollary 4.6.3

By hypothesis, $Q_1 \sqsubseteq^I Q_2$ $(\underline{\mathbf{1}})$ holds. By Lemma 3.6.2 (page 114) on (1), there exists a renaming ϕ disjoint from I and a prefix Q_0 such that we have $Q_1 \sqsubseteq \phi(Q_2)Q_0$ and $\phi(Q_2)Q_0 \equiv^I Q_2$ $(\underline{\mathbf{2}})$. By Lemma 4.6.2, `unify` (Q_1, τ_1, τ_2) succeeds with Q'_1 and we have $Q'_1 \sqsubseteq^{\mathsf{dom}(Q_1)} \phi(Q_2)Q_0$. By Property 3.4.2.i (page 106) and (2), we get $Q'_1 \sqsubseteq^I Q_2$. This is the expected result. ■

Proof of Lemma 6.1.4

This property is shown by induction on the the derivation of $(Q)\ \Gamma \vdash a : \sigma$. Case Var is immediate. Cases App, Inst and Oracle are by induction hypothesis.

◦ Case Fun: The premise is $(Q)\ \Gamma, x : \tau_0 \vdash a_0 : \tau$ and σ is $\tau_0 \to \tau$. By alpha-conversion of the lambda-bound variable x, we can freely assume that x is not bound in Γ'. Hence, by induction hypothesis, we have $(Q)\ \Gamma, x : \tau_0, \Gamma' \vdash a_0 : \tau$, which is identified with $(Q)\ \Gamma, \Gamma', x : \tau_0 \vdash a_0 : \tau$. Hence, by Rule Fun, we get $(Q)\ \Gamma, \Gamma' \vdash \lambda(x)\ a_0 : \tau_0 \to \tau$, which is the expected result.

◦ Case Let: We have $(Q)\ \Gamma \vdash a_1 : \sigma_1$ and $(Q)\ \Gamma, x : \sigma_1 \vdash a_2 : \sigma$. By induction hypothesis, we have $(Q)\ \Gamma, \Gamma' \vdash a_1 : \sigma_1$ and $(Q)\ \Gamma, x : \sigma_1, \Gamma' \vdash a_2 : \sigma$, which can be written $(Q)\ \Gamma, \Gamma', x : \sigma_1 \vdash a_2 : \sigma$. Hence, by Rule Let, we get the expected result.

◦ Case Gen: The premise is $(Q, \alpha \diamond \sigma_a)\ \Gamma \vdash a : \sigma'$ $(\underline{\mathbf{1}})$, and σ is $\forall\,(\alpha \diamond \sigma_a)\ \sigma'$. Besides, $\alpha \notin \mathsf{ftv}(\Gamma)$. Let α' be a fresh variable (that is, not in $\mathsf{ftv}(\Gamma, \Gamma')$). By Lemma 6.1.1 applied to (1) with the renaming $[\alpha'/\alpha]$, we get $(Q, \alpha' \diamond \sigma_a)\ \Gamma \vdash a : \sigma'[\alpha'/\alpha]$, and this derivation has the same size as (1). Hence, by induction hypothesis, we have a derivation of $(Q, \alpha' \diamond \sigma_a)\ \Gamma, \Gamma' \vdash a : \sigma'[\alpha'/\alpha]$. Applying Gen, we get $(Q)\ \Gamma, \Gamma' \vdash a : \forall\,(\alpha' \diamond \sigma_a)\ \sigma'[\alpha'/\alpha]$, that is (by alpha-conversion) $(Q)\ \Gamma, \Gamma' \vdash a : \sigma$, which is the expected result. ■

Proof of Lemma 6.2.1

It suffices to show that each rule defining $\vdash^{\triangledown}$ is derivable in $\vdash$.

◦ Case Var$^{\triangledown}$: This rule is identical to Rule Var.

◦ Case Fun$^{\triangledown}$: We choose α fresh, that is $\alpha \notin \mathsf{dom}(QQ') \cup \mathsf{ftv}(\Gamma, \tau_0, \sigma)$. By hypothesis, $(QQ')\ \Gamma, x : \tau_0 \vdash a : \sigma$ holds, and $\mathsf{dom}(Q') \# \mathsf{ftv}(\Gamma)$. By Corollary 6.1.3 (page 153), we get $(QQ', \alpha \geq \sigma)\ \Gamma, x : \tau_0 \vdash a : \sigma$. Hence, the following derivation:

$$\dfrac{\dfrac{\dfrac{(QQ', \alpha \geq \sigma)\ \Gamma, x : \tau_0 \vdash a : \sigma \qquad \dfrac{}{(QQ', \alpha \geq \sigma)\ \sigma \sqsubseteq \alpha}\ \textsc{I-Hyp}}{(QQ', \alpha \geq \sigma)\ \Gamma, x : \tau_0 \vdash a : \alpha}\ \textsc{Inst}}{(QQ', \alpha \geq \sigma)\ \Gamma \vdash \lambda(x)\ a : \tau_0 \to \alpha}\ \textsc{Fun}}{(Q)\ \Gamma \vdash \lambda(x)\ a : \forall\,(Q', \alpha \geq \sigma)\ \tau_0 \to \alpha}\ \textsc{Gen}$$

∘ Case App$^{\triangledown}$: We have already shown that App* is derivable. Then App$^{\triangledown}$ can be derived with Inst and App*.

∘ Case Let$^{\triangledown}$ We choose α fresh, and Q_2 is an abbreviation for $(Q, \alpha \geq \sigma_2)$. By hypothesis, we have $(Q)\ \Gamma \vdash a_1 : \sigma_1$ and $(Q)\ \Gamma, x : \sigma_1 \vdash a_2 : \sigma_2$. By Corollary 6.1.3 (page 153), we get $(Q_2)\ \Gamma \vdash a_1 : \sigma_1$ and $(Q_2)\ \Gamma, x : \sigma_1 \vdash a_2 : \sigma_2$. Then we have the following derivation:

$$\dfrac{\dfrac{(Q_2)\ \Gamma \vdash a_1 : \sigma_1 \qquad \dfrac{\begin{array}{c}(Q_2)\ \Gamma, x : \sigma_1 \vdash a_2 : \sigma_2\\ (Q_2)\ \sigma_2 \sqsubseteq \alpha\end{array}}{(Q_2)\ \Gamma, x : \sigma_1 \vdash a_2 : \alpha}\,\textsc{Inst}}{(Q_2)\ \Gamma \vdash \texttt{let}\ x = a_1\ \texttt{in}\ a_2 : \alpha}\,\textsc{Let}}{(Q)\ \Gamma \vdash \texttt{let}\ x = a_1\ \texttt{in}\ a_2 : \sigma_2}\,\textsc{Gen+Inst}$$

∘ Case Oracle$^{\triangledown}$ is a combination of Oracle and Inst. ■

Proof of Lemma 6.2.3

By induction on the derivation of $(Q)\ \Gamma \vdash' a : \sigma$. Cases App$^{\triangledown}$, Fun$^{\triangledown}$, Let$^{\triangledown}$, Inst and Oracle$^{\triangledown}$ are by induction hypothesis.

∘ Case Var$^{\triangledown}$: We have $(Q)\ \Gamma \vdash' z : \sigma$ and $z : \sigma$ is in Γ. By definition, $z : \sigma'$ is in Γ', with $(Q)\ \sigma' \sqsubseteq \sigma$. Hence, we have (Rule Var$^{\triangledown}$) $(Q)\ \Gamma' \vdash' z : \sigma'$ and we get the expected result $(Q)\ \Gamma' \vdash' z : \sigma$ by Rule Inst.

∘ Case Gen: The premises are $(Q, \alpha \diamond \sigma_a)\ \Gamma \vdash' a : \sigma'$ **($\underline{1}$)** and $\alpha \notin \mathsf{ftv}(\Gamma)$. The conclusion is $(Q)\ \Gamma \vdash' a : \forall\,(\alpha \diamond \sigma_a)\ \sigma'$. We have $(Q)\ \Gamma' \sqsubseteq \Gamma$. By well-formedness, all free type variables of Γ' must be bound in Q. Since $\alpha \notin \mathsf{dom}(Q)$, we have $\alpha \notin \mathsf{ftv}(\Gamma')$. The result is by induction hypothesis on (1) and Rule Gen, then. ■

Proof of Property 6.2.5

Property i: By hypothesis, we have a derivation ending with Gen. Its premise is $(Q, \alpha \diamond \sigma)\ \Gamma \vdash' a : \sigma'$ **($\underline{1}$)** and its conclusion is $(Q)\ \Gamma \vdash' a : \forall\,(\alpha \diamond \sigma)\ \sigma'$ **($\underline{2}$)**. Besides, $\alpha \notin \mathsf{ftv}(\Gamma)$ **($\underline{3}$)**. We have to show that there exists a derivation of $(Q)\ \Gamma \vdash' a : \sigma''$ **($\underline{4}$)**, with $\sigma'' \equiv \forall\,(\alpha \diamond \sigma)\ \sigma'$ **($\underline{5}$)**, and the size of (4) must be strictly smaller than the size of (2), or, equivalently, the size of (4) is smaller than or equal to the size of (1). The proof is by induction on the size of the term, then on the size of the derivation, and then by case on the penultimate rule (the last rule being, by hypothesis, Rule Gen).

∘ Case Var$^{\triangledown}$: The premise of this penultimate rule is $x : \sigma' \in \Gamma$ **($\underline{6}$)**, and its conclusion is (1), with a being x. We have $\alpha \notin \mathsf{ftv}(\sigma')$ by (3) and (6). Thus $\forall\,(\alpha \diamond \sigma)\ \sigma' \equiv \sigma'$ **($\underline{7}$)** holds by Eq-Free. We have $(Q)\ \Gamma \vdash' x : \sigma'$ **($\underline{8}$)** by Var$^{\triangledown}$, that is (4), taking $\sigma'' = \sigma'$. As expected (5) holds by (7). Besides, the size of (8) is one, which is equal to the size of (1).

∘ Case Fun$^\triangledown$: The premise is $(Q, \alpha \diamond \sigma, Q')\ \Gamma, x : \tau_0 \vdash' a_0 : \sigma_0$ $\mathbf{(\underline{9})}$ and $\mathsf{dom}(Q') \# \mathsf{ftv}(\Gamma)$ $\mathbf{(\underline{10})}$. Besides, σ' is $\forall\,(Q', \beta \geq \sigma_0)\ \tau_0 \to \beta$ $\mathbf{(\underline{11})}$. Hence, (1) is written $(Q, \alpha \diamond \sigma)\ \Gamma \vdash' \lambda(x)\ a_0 : \forall\,(Q', \beta \geq \sigma_0)\ \tau_0 \to \beta$. By (3) and (10), we have $\mathsf{dom}(\alpha \diamond \sigma, Q') \# \mathsf{ftv}(\Gamma)$. Hence, by Rule Fun$^\triangledown$ applied to (9), we get $(Q)\ \Gamma \vdash' \lambda(x)\ a_0 : \forall\,(\alpha \diamond \sigma, Q', \beta \geq \sigma_0)\ \tau_0 \to \beta$ $\mathbf{(\underline{12})}$. Taking $\sigma'' = \forall\,(\alpha \diamond \sigma, Q', \beta \geq \sigma_0)\ \tau_0 \to \beta$, we have shown (4) and (5), using (11). Additionally, the size of (12) is the size of (1).

∘ Case App$^\triangledown$: The premises are $(Q, \alpha \diamond \sigma)\ \Gamma \vdash' a_1 : \sigma_1$ $\mathbf{(\underline{13})}$ and $(Q, \alpha \diamond \sigma)\ \Gamma \vdash' a_2 : \sigma_2$ $\mathbf{(\underline{14})}$. Moreover, we have $(Q, \alpha \diamond \sigma)\ \sigma_1 \sqsubseteq \forall\,(Q')\ \tau_2 \to \tau_1$ $\mathbf{(\underline{15})}$ and $(Q, \alpha \diamond \sigma)\ \sigma_2 \sqsubseteq \forall\,(Q')\ \tau_2$ $\mathbf{(\underline{16})}$. Besides, σ' is $\forall\,(Q')\ \tau_1$. By (3), (13) and Rule Gen, we get $(Q)\ \Gamma \vdash' a_1 : \forall\,(\alpha \diamond \sigma)\ \sigma_1$ $\mathbf{(\underline{17})}$. Besides, the size of (17) is the size of (13) plus one. By induction hypothesis, $(Q)\ \Gamma \vdash' a_1 : \sigma'_1$ $\mathbf{(\underline{18})}$ is derivable with $\sigma'_1 \equiv \forall\,(\alpha \diamond \sigma)\ \sigma_1$ $\mathbf{(\underline{19})}$. Moreover, the size of (18) is strictly smaller than the size of (17), that is, smaller than or equal to the size of (13). Similarly, $(Q)\ \Gamma \vdash' a_2 : \sigma'_2$ $\mathbf{(\underline{20})}$ is derivable with $\sigma'_2 \equiv \forall\,(\alpha \diamond \sigma)\ \sigma_2$ $\mathbf{(\underline{21})}$, and the size of (20) is smaller than or equal to the size of (14). By (19), (15) and R-Context-R we get $(Q)\ \sigma'_1 \sqsubseteq \forall\,(\alpha \diamond \sigma, Q')\ \tau_2 \to \tau_1$ $\mathbf{(\underline{22})}$. Similarly, (21) and (16) give $(Q)\ \sigma'_2 \sqsubseteq \forall\,(\alpha \diamond \sigma, Q')\ \tau_2$ $\mathbf{(\underline{23})}$. By (18), (20), (22), and (23), and Rule App$^\triangledown$, we get $(Q)\ \Gamma \vdash' a_1\ a_2 : \forall\,(\alpha \diamond \sigma, Q')\ \tau_1$ $\mathbf{(\underline{24})}$. This corresponds to (4). Additionally, the size of (24) is one plus the size of (18) and the size of (20). Hence, it is smaller than or equal to one plus the size of (13) plus the size of (14), that is, the size of (1).

∘ Case Inst: The premises are $(Q, \alpha \diamond \sigma)\ \Gamma \vdash' a : \sigma_0$ $\mathbf{(\underline{25})}$. and $(Q, \alpha \diamond \sigma)\ \sigma_0 \sqsubseteq \sigma'$ $\mathbf{(\underline{26})}$. By Rule Gen, $(Q)\ \Gamma \vdash' a : \forall\,(\alpha \diamond \sigma)\ \sigma_0$ holds. By induction hypothesis, $(Q)\ \Gamma \vdash' a : \sigma'_0$ $\mathbf{(\underline{27})}$ holds, with $\sigma'_0 \equiv \forall\,(\alpha \diamond \sigma)\ \sigma_0$ $\mathbf{(\underline{28})}$, and the size of (27) is smaller than or equal to the size of (25). From (26), R-Context-R, and (28), we get $(Q)\ \sigma'_0 \sqsubseteq \forall\,(\alpha \diamond \sigma)\ \sigma'$. Hence, with (27) and Inst, this gives $(Q)\ \Gamma \vdash' a : \forall\,(\alpha \diamond \sigma)\ \sigma'$ $\mathbf{(\underline{29})}$. Besides, the size of (29) is smaller than or equal to one plus the size of (25). Hence, it is smaller than or equal to the size of (1).

∘ Case Oracle$^\triangledown$ is similar.

∘ Case Gen: The premise is $(Q, \alpha \diamond \sigma, \alpha_1 \diamond_1 \sigma_1)\ \Gamma \vdash' a : \sigma_0$ $\mathbf{(\underline{30})}$, and σ' is $\forall\,(\alpha_1 \diamond_1 \sigma_1)\ \sigma_0$. By induction hypothesis, we have a derivation of $(Q, \alpha \diamond \sigma)\ \Gamma \vdash' a : \sigma'_0$ $\mathbf{(\underline{31})}$ which is smaller than or equal to the size of (30) and with $\sigma'_0 \equiv \sigma'$ $\mathbf{(\underline{32})}$. By Rule Gen, we get $(Q)\ \Gamma \vdash' a : \forall\,(\alpha \diamond \sigma)\ \sigma'_0$. By induction hypothesis, we have a derivation of $(Q)\ \Gamma \vdash' a : \sigma''$ $\mathbf{(\underline{33})}$, where $\sigma'' \equiv \forall\,(\alpha \diamond \sigma)\ \sigma'_0$ $\mathbf{(\underline{34})}$, and the size of (33) is smaller than or equal to the size of (31). We note that $\sigma'' \equiv \forall\,(\alpha \diamond \sigma)\ \sigma'$ by (32) and (34). Besides, the size of (33) is smaller than or equal to the size of (30). Thus, it is strictly smaller than the size of (1).

∘ Case Let$^\triangledown$: The premises are $(Q, \alpha \diamond \sigma)\ \Gamma \vdash' a_1 : \sigma_1$ $\mathbf{(\underline{35})}$ as well as $(Q, \alpha \diamond \sigma)\ \Gamma, x : \sigma_1 \vdash' a_2 : \sigma'$ $\mathbf{(\underline{36})}$. By Gen and (35), we get $(Q)\ \Gamma \vdash' a_1 : \forall\,(\alpha \diamond \sigma)\ \sigma_1$. By induction hypothesis, we have a derivation of $(Q)\ \Gamma \vdash' a_1 : \sigma'_1$ $(\underline{\mathbf{37}})$, with $\sigma'_1 \equiv \forall\,(\alpha \diamond \sigma)\ \sigma_1$ $(\underline{\mathbf{38}})$. Besides, the size of (37) is smaller than or equal to the size of (35). We note that $(Q, \alpha \diamond \sigma)\ \forall\,(\alpha \diamond \sigma)\ \sigma_1 \sqsubseteq \sigma_1$ $(\underline{\mathbf{39}})$ is derivable by I-Drop*. From (39) and (38), we get $(Q, \alpha\diamond\sigma)\ \sigma'_1 \sqsubseteq \sigma_1$ $(\underline{\mathbf{40}})$. By Strengthen’, (36), and (40), we get $(Q, \alpha\diamond\sigma)\ \Gamma, x : \sigma'_1 \vdash' a_2 : \sigma'$ $(\underline{\mathbf{41}})$. Note that we have $\alpha \notin \mathsf{ftv}(\sigma)$ by well formedness of the prefix of (35). Hence, we have $\alpha \notin \mathsf{ftv}(\forall\,(\alpha\diamond\sigma)\ \sigma_1)$, which implies $\alpha \notin \mathsf{ftv}(\sigma'_1)$ $(\underline{\mathbf{42}})$ by Property 1.5.4.iii (page 50) and (38). By Rule Gen, (42) and (41), we get $(Q)\ \Gamma, x : \sigma'_1 \vdash' a_2 : \forall\,(\alpha \diamond \sigma)\ \sigma'$ $(\underline{\mathbf{43}})$. By Rule Let$^\triangledown$, (37) and (43), we get $(Q)\ \Gamma \vdash' a : \forall\,(\alpha \diamond \sigma)\ \sigma'$ $(\underline{\mathbf{44}})$. The size of (44) is one plus the size of (37) (remember that the right premise of Let$^\triangledown$ is not counted). Hence, the size of (44) is smaller than or equal to one plus the size of (35). Consequently, the size of (44) is smaller than or equal to the size of (1).

Property ii: By induction on the size of the term, then on the size of the derivation. Case Var$^\triangledown$ is immediate. Cases Fun$^\triangledown$, App$^\triangledown$, Inst, and Oracle$^\triangledown$ are by induction hypothesis.

∘ Case Let$^\triangledown$: The premises are $(Q)\ \Gamma \vdash' a_1 : \sigma_1$ and $(Q)\ \Gamma, x : \sigma_1 \vdash' a_2 : \sigma_2$ $(\underline{\mathbf{1}})$. By induction hypothesis, $(Q)\ \Gamma \vdash' a_1 : \sigma'_1$ $(\underline{\mathbf{2}})$ is derivable not using Gen, with $\sigma'_1 \equiv \sigma_1$. By Strengthen’ and (1), we get $(Q)\ \Gamma, x : \sigma'_1 \vdash' a_2 : \sigma_2$. By induction hypothesis (a_2 is a subexpression of a), we have a derivation of $(Q)\ \Gamma, x : \sigma'_1 \vdash' a_2 : \sigma'_2$ $(\underline{\mathbf{3}})$ not using Gen and $\sigma'_2 \equiv \sigma_2$. By Let$^\triangledown$, (2), and (3), we have a derivation of $(Q)\ \Gamma \vdash' a : \sigma'_2$ not using Gen, and $\sigma'_2 \equiv \sigma_2$. This is the expected result.

∘ Case Gen: By Property i, there exists a strictly smaller derivation not ending with Gen of $(Q)\ \Gamma \vdash' a : \sigma''_a$ with $\sigma''_a \equiv \sigma_a$ $(\underline{\mathbf{4}})$. Hence, by induction hypothesis, we have a derivation not using Gen of $(Q)\ \Gamma \vdash' a : \sigma'_a$ with $\sigma'_a \equiv \sigma''_a$ $(\underline{\mathbf{5}})$. By R-Trans, (5), and (4), we have $\sigma'_a \equiv \sigma_a$, which is the expected result.

Property iii: By induction on the size of the term and on the derivation.

∘ Case Var$^\triangledown$ is immediate.

∘ Case Inst: The premises are $(Q)\ \Gamma \vdash' a : \sigma$ $(\underline{\mathbf{1}})$ and $(Q)\ \sigma \sqsubseteq \sigma_a$ $(\underline{\mathbf{2}})$. We get the expected result by induction hypothesis on (1) and (2).

∘ Case Gen: By hypothesis, this case cannot occur.

∘ Case Fun$^\triangledown$: The premise is $(QQ')\ \Gamma, x : \tau_0 \vdash^{\triangledown} a : \sigma$ and $\mathsf{dom}(Q') \cap \mathsf{ftv}(\Gamma) = \emptyset$. By induction hypothesis, we have a derivation of $(QQ')\ \Gamma, x : \tau_0 \vdash^{\triangledown} a : \sigma'$ (not using Inst) such that $(QQ')\ \sigma' \sqsubseteq \sigma$ $(\underline{\mathbf{3}})$. By Rule Fun$^\triangledown$, we have $(Q)\ \Gamma \vdash^{\triangledown} \lambda(x)\ a : \forall\,(Q', \alpha \geq \sigma')\ \tau_0 \to \alpha$, and we easily check that $(Q)\ \forall\,(Q', \alpha \geq \sigma')\ \tau_0 \to \alpha \sqsubseteq \forall\,(Q', \alpha \geq \sigma)\ \tau_0 \to \alpha$ holds by rules R-Context-R, R-Context-Flexible, and (3).

∘ Case Oracle$^\triangledown$ and App$^\triangledown$: These rules already contains an occurrence of Inst by construction. Thus, the we get the result by induction hypothesis.

∘ Case Let$^\triangledown$ The premises are $(Q)\ \Gamma \vdash' a_1 : \sigma_1$ and $(Q)\ \Gamma, x : \sigma_1 \vdash' a_2 : \sigma_2$. By induction hypothesis, we have a derivation of $(Q)\ \Gamma \vdash^{\triangledown} a_1 : \sigma'_1$ $(\underline{\mathbf{4}})$ such that

$(Q)\ \sigma'_1 \sqsubseteq \sigma_1$ holds. Hence, by Rule STRENGTHEN, we have $(Q)\ \Gamma, x : \sigma'_1 \vdash' a_2 : \sigma_2$ **(5)**. Note that a_2 is a subterm of a, thus the induction hypothesis can be applied to (5). Hence, by induction hypothesis, we have a derivation of $(Q)\ \Gamma, x : \sigma'_1 \vdash^{\triangledown} a_2 : \sigma'_2$ with $(Q)\ \sigma'_2 \sqsubseteq \sigma_2$ **(6)**. We conclude with Rule LET$^{\triangledown}$, (4), and (6). ■

Proof of Property 8.1.5

Property i: It is shown by structural induction on σ.

Property ii: We prove the first rule by induction on the size of Q. If Q is $\emptyset$, $\langle\!\langle\emptyset\rangle\!\rangle$ is $\emptyset, \mathtt{id}$, and $\langle\!\langle\sigma\rangle\!\rangle = \forall\,(\emptyset)\ \mathtt{id}(\langle\!\langle\sigma\rangle\!\rangle)$ holds. Otherwise, Q is $(\alpha \diamond \sigma_0, Q')$. Let $\forall\,(Q_1)\ \tau_1$ be $\langle\!\langle\sigma_0\rangle\!\rangle$ and Q'_1, θ_1 be $\langle\!\langle Q'\rangle\!\rangle$. By induction hypothesis, $\langle\!\langle\forall\,(Q')\ \sigma\rangle\!\rangle = \forall\,(Q'_1)\ \theta_1(\langle\!\langle\sigma\rangle\!\rangle)$. Hence, by Definition 8.1.4, $\langle\!\langle\forall\,(Q)\ \sigma\rangle\!\rangle$ is $\forall\,(Q_1Q'_1)\ \theta_1(\langle\!\langle\sigma\rangle\!\rangle)[\tau_1/\alpha]$ **(1)**. By definition, $\langle\!\langle Q\rangle\!\rangle$ is $Q_1Q'_1, [\tau_1/\alpha] \circ \theta_1$. Hence, (1) is the expected result.

The second rule is proved by induction on the size of Q_1. If Q_1 is $\emptyset$, the result is immediate. Otherwise, Q_1 is $(\alpha \diamond \sigma, Q'_1)$. Let Q''_1, θ'_1 be $\langle\!\langle Q'_1\rangle\!\rangle$. By induction hypothesis, $\langle\!\langle Q'_1Q_2\rangle\!\rangle$ is $Q''_1Q'_2, \theta'_1 \circ \theta_2$. Let $\forall\,(Q)\ \tau$ be $\langle\!\langle\sigma\rangle\!\rangle$. By definition, $\langle\!\langle Q_1Q_2\rangle\!\rangle$ is $QQ''_1Q'_2, [\tau/\alpha] \circ \theta'_1 \circ \theta_2$. We get the expected result by observing that $\langle\!\langle Q_1\rangle\!\rangle$ is $QQ''_1, [\tau/\alpha] \circ \theta'_1$.

Property iii: It is shown by structural induction on σ.

Property iv: By hypothesis, we have $\sigma \equiv \tau$. By Property 1.5.4.i (page 50), we have $\sigma/ = \tau/$. By Property iii, we have $\sigma/ = \langle\!\langle\sigma\rangle\!\rangle/$. Hence, $\langle\!\langle\sigma\rangle\!\rangle/ = \tau/$. Moreover, $\langle\!\langle\sigma\rangle\!\rangle$ is necessarily of the form $\forall\,(\bar\alpha)\ \tau'$. For any u in $\mathsf{dom}(\langle\!\langle\sigma\rangle\!\rangle)$, we have $\langle\!\langle\sigma\rangle\!\rangle/u = \tau/u$, hence $\langle\!\langle\sigma\rangle\!\rangle/u \neq \bot$, that is, $\forall\,(\bar\alpha)\ \tau'/u \neq \bot$ for any u in $\mathsf{dom}(\forall\,(\bar\alpha)\ \tau')$. Hence, we must have $\bar\alpha \mathrel{\#} \mathsf{ftv}(\tau')$. Consequently, $\forall\,(\bar\alpha)\ \tau'/ = \tau'/$, thus $\tau'/ = \tau/$. This implies $\tau = \tau'$. We have shown that $\langle\!\langle\sigma\rangle\!\rangle$ is $\forall\,(\bar\alpha)\ \tau$ with $\bar\alpha \mathrel{\#} \mathsf{ftv}(\tau)$.

Property v: It is shown by induction on the size of Q. If Q is $\emptyset$, the result is immediate. Otherwise, Q is $(Q', \alpha \diamond \sigma)$. Let $\forall\,(Q_1)\ \tau_1$ be $\langle\!\langle\sigma\rangle\!\rangle$, and Q'_1, θ_1 be $\langle\!\langle Q'\rangle\!\rangle$. By induction hypothesis, θ_1 is of the form $\theta'_1 \circ \widehat{Q'}$. By definition, θ is $[\tau_1/\alpha] \circ \theta_1$, that is $[\tau_1/\alpha] \circ \theta'_1 \circ \widehat{Q'}$. If $\sigma \notin \mathcal{T}$, then $\widehat{Q} = \widehat{Q'}$, and θ is $[\tau_1/\alpha] \circ \theta'_1 \circ \widehat{Q}$. This is the expected result. Otherwise, $\sigma \equiv \tau$, and τ_1 is τ by Property iv. Moreover, $\widehat{Q}$ is $[\tau/\alpha] \circ \widehat{Q'}$. Then θ is $[\tau/\alpha] \circ \theta'_1 \circ \widehat{Q'}$. Consequently, θ is equivalent to $[\tau/\alpha] \circ \theta'_1 \circ [\tau/\alpha] \circ \widehat{Q'}$, that is, $[\tau/\alpha] \circ \theta'_1 \circ \widehat{Q}$. This is the expected result.

Property vi: It is by induction on the derivation of $(Q)\ \sigma_1 \sqsubseteq \sigma_2$. Cases R-TRANS, A-HYP, ... and I-RIGID are not possible since the derivation is flexible.

- CASE EQ-REFL: immediate.
- CASE R-TRANS: By induction.

∘ Case R-Trans: By induction.

∘ Case Eq-Free: This case corresponds to adding or removing useless binders in an ML type scheme.

∘ Case Eq-Comm: This case corresponds to commuting binders in an ML type scheme.

∘ Case Eq-Var: $\langle\!\langle \forall\,(\alpha \geq \sigma)\ \alpha \rangle\!\rangle$ and $\langle\!\langle \sigma \rangle\!\rangle$ are identical.

∘ Case R-Context-R: Let Q_a be $(Q, \alpha \geq \sigma_0)$. Let $\forall\,(Q_0)\ \tau_0$ be $\langle\!\langle \sigma_0 \rangle\!\rangle$. Let Q', θ be $\langle\!\langle Q \rangle\!\rangle$, and let θ' be $[\tau_0/\alpha]$. The premise is $(Q_a)\ \sigma'_1 \sqsubseteq \sigma'_2$, and we have $\sigma_1 = \forall\,(\alpha{\geq}\sigma_0)\ \sigma'_1$, as well as $\sigma_2 = \forall\,(\alpha{\geq}\sigma_0)\ \sigma'_2$. We have $\langle\!\langle Q_a \rangle\!\rangle = Q'Q_0, \theta \circ \theta'$ by Property ii. By induction hypothesis, we have $\theta \circ \theta'(\langle\!\langle \sigma'_1 \rangle\!\rangle) \sqsubseteq_{ML} \theta \circ \theta'(\langle\!\langle \sigma'_2 \rangle\!\rangle)$ (**1**). Hence, we have

$$\begin{array}{rcll}
\theta(\langle\!\langle \forall\,(\alpha \geq \sigma_0)\ \sigma_1 \rangle\!\rangle) & = & \theta(\forall\,(Q_0)\ \theta'(\langle\!\langle \sigma_1 \rangle\!\rangle) & \text{by definition} \\
& = & \forall\,(Q_0)\ \theta \circ \theta'(\langle\!\langle \sigma_1 \rangle\!\rangle) & \\
& \sqsubseteq_{ML} & \forall\,(Q_0)\ \theta \circ \theta'(\langle\!\langle \sigma_2 \rangle\!\rangle) & \text{by (1)} \\
& = & \theta(\forall\,(Q_0)\ \theta'(\langle\!\langle \sigma_2 \rangle\!\rangle)) & \\
& = & \theta(\langle\!\langle \forall\,(\alpha \geq \sigma_0)\ \sigma_2 \rangle\!\rangle) & \text{by definition}
\end{array}$$

This is the expected result.

∘ Case R-Context-L: See R-Context-Flexible

∘ Case R-Context-Flexible: The premise is $\forall\,(Q)\ \sigma'_1 \sqsubseteq \sigma'_2$, and the conclusion is $\forall\,(Q)\ \forall\,(\alpha{\geq}\sigma'_1)\ \sigma \sqsubseteq \forall\,(\alpha{\geq}\sigma'_2)\ \sigma$. Let $Q'\theta$ be $\langle\!\langle Q \rangle\!\rangle$. Let $\forall\,(Q_1)\ \tau_1$ be $\langle\!\langle \sigma'_1 \rangle\!\rangle$ and $\forall\,(Q_2)\ \tau_2$ be $\langle\!\langle \sigma'_2 \rangle\!\rangle$. Let θ_1 be $[\tau_1/\alpha]$ and θ_2 be $[\tau_2/\alpha]$. We have $\langle\!\langle \forall\,(\alpha \geq \sigma'_1)\ \sigma \rangle\!\rangle = \forall\,(Q_1)\ \theta_1(\langle\!\langle \sigma \rangle\!\rangle)$ and $\langle\!\langle \forall\,(\alpha{\geq}\sigma'_2)\ \sigma \rangle\!\rangle = \forall\,(Q_2)\ \theta_2(\langle\!\langle \sigma \rangle\!\rangle)$. By induction hypothesis, we have $\theta(\langle\!\langle \sigma'_1 \rangle\!\rangle) \sqsubseteq_{ML} \theta(\langle\!\langle \sigma'_2 \rangle\!\rangle)$, that is, $\forall\,(Q_1)\ \theta(\tau_1) \sqsubseteq_{ML} \forall\,(Q_2)\ \theta(\tau_2)$. By Lemma 8.1.1 (page 171), we get $\forall\,(Q_1)\ \theta \circ \theta_1(\langle\!\langle \sigma \rangle\!\rangle) \sqsubseteq_{ML} \forall\,(Q_2)\ \theta \circ \theta_2(\langle\!\langle \sigma \rangle\!\rangle)$, which is the expected result.

∘ Case Eq-Mono: The premises are $(\alpha \diamond \sigma_0) \in Q$ and $(Q)\ \sigma_0 \equiv \tau_0$. The conclusion is $(Q)\ \tau \equiv \tau[\tau_0/\alpha]$. We have $\widehat{Q}(\alpha) = \widehat{Q}(\tau_0)$ by definition. Let Q', θ be $\langle\!\langle Q \rangle\!\rangle$. We have $\theta(\alpha) = \theta(\tau_0)$ by Property v. Hence, we have $\theta(\tau) = \theta(\tau[\tau_0/\alpha])$. This is the expected result.

∘ Case I-Hyp: We have $(\alpha_1 \geq \sigma_1) \in Q$ and the conclusion is $(Q)\ \sigma_1 \sqsubseteq \alpha_1$. Let $\forall\,(Q_1)\ \tau_1$ be $\langle\!\langle \sigma_1 \rangle\!\rangle$ and Q', θ be $\langle\!\langle Q \rangle\!\rangle$. By definition, $\theta(\alpha_1) = \theta(\tau_1)$. Besides, $\theta(\forall\,(Q_1)\ \tau_1)$ is $\forall\,(Q_1)\ \theta(\tau_1)$ since Q_1 is unconstrained, and $\forall\,(Q_1)\ \theta(\tau_1) \sqsubseteq_{ML} \theta(\tau_1)$ holds, that is, $\forall\,(Q_1)\ \theta(\tau_1) \sqsubseteq_{ML} \theta(\alpha_1)$. Thus we have the expected result $\theta(\forall\,(Q_1)\ \tau_1) \sqsubseteq_{ML} \theta(\alpha_1)$.

∘ Case I-Bot: $\forall\,(\alpha)\ \alpha \sqsubseteq_{ML} \sigma$ holds for any ML type σ.

This proves the property. ■

Proof of Lemma 8.3.2

Let us name the conclusions (1) through (4) in order of appearance:

$$(Q_0)\ \Gamma \vdash a : \forall\,(Q')\ \sigma'_1\ (\underline{\mathbf{1}}) \qquad Q_0Q \sqsubseteq Q_0Q'\ (\underline{\mathbf{2}}) \qquad (Q_0Q')\ \sigma'_1 \mathrel{-\!\!\!\sqsupset} \sigma\ (\underline{\mathbf{3}})$$

$$(Q_0)\ \forall\,(Q')\ \sigma \sqsubseteq \sigma_0\ (\underline{\mathbf{4}})$$

We write Q'' for $(Q, \alpha_1 = \sigma, \alpha_0 = \sigma)$. By Lemma 6.2.6, it suffices to show the result for $(Q_0)\ \Gamma \vdash^{\triangledown} (a : \exists\,(Q)\ \sigma) : \forall\,(Q_1)\ \tau_0$ $\mathbf{(\underline{5})}$ with $(Q_0)\ \forall\,(Q_1)\ \tau_0 \sqsubseteq \sigma_0$ $\mathbf{(\underline{6})}$. The premises of (5) are necessarily

$$(Q_0)\ \Gamma \vdash^{\triangledown} a : \sigma_1 \quad (\underline{\mathbf{7}}) \qquad\qquad (Q_0)\ \sigma_1 \sqsubseteq \forall\,(Q_1)\ \tau_1 \quad (\underline{\mathbf{8}})$$

$$(Q_0)\ \forall\,(Q'')\ \alpha_1 \to \alpha_0 \sqsubseteq \forall\,(Q_1)\ \tau_1 \to \tau_0 \quad (\underline{\mathbf{9}})$$

∘ Soundness Assume (1), (2), (3), and (4). Take $(Q', \alpha_1 = \sigma'_1, \alpha_0 = \sigma)$ for Q_1, α_1 for τ_1, and α_0 for τ_0. By Lemma 6.2.6 and (1), there exists σ_1 such that (7) holds and $(Q_0)\ \sigma_1 \sqsubseteq \forall\,(Q')\ \sigma'_1$ $\mathbf{(\underline{10})}$ holds. By Eq-Var*, we have $(Q_0)\ \forall\,(Q_1)\ \tau_1 \equiv \forall\,(Q')\ \sigma'_1$. Hence, (10) gives $(Q_0)\ \sigma_1 \sqsubseteq \forall\,(Q_1)\ \tau_1$ (8). We have $(Q_0Q)\ \forall\,(Q'')\ \alpha_1 \to \alpha_0 \sqsubseteq \forall\,(\alpha_1 = \sigma)\ \forall\,(\alpha_0 = \sigma)\ \alpha_1 \to \alpha_0$ $\mathbf{(\underline{11})}$ by I-Drop*. By (2) and Lemma 3.6.4 applied to (11), we get $(Q_0Q')\ \forall\,(Q'')\ \alpha_1 \to \alpha_0 \sqsubseteq \forall\,(\alpha_1 = \sigma)\ \forall\,(\alpha_0 = \sigma)\ \alpha_1 \to \alpha_0$. Using I-Abstract with (3), we get $(Q_0Q')\ \forall\,(Q'')\ \alpha_1 \to \alpha_0 \sqsubseteq \forall\,(\alpha_1 = \sigma'_1)\ \forall\,(\alpha_0 = \sigma)\ \alpha_1 \to \alpha_0$. Hence, by R-Context-R and Eq-Free, we get $(Q_0)\ \forall\,(Q'')\ \alpha_1 \to \alpha_0 \sqsubseteq \forall\,(Q')\ \forall\,(\alpha_1 = \sigma'_1)\ \forall\,(\alpha_0 = \sigma)$ $\alpha_1 \to \alpha_0$, that is $(Q_0)\ \forall\,(Q'')\ \alpha_1 \to \alpha_0 \sqsubseteq \forall\,(Q_1)\ \alpha_1 \to \alpha_0$ (9). The premises (7), (8), (9) are checked, thus Rule App applies and gives $(Q_0)\ \Gamma \vdash^{\triangledown} (a : \exists\,(Q)\ \sigma) : \forall\,(Q_1)\ \tau_0$ (5). Finally, $(Q_0)\ \forall\,(Q_1)\ \tau_0 \equiv \forall\,(Q')\ \sigma$ holds by Eq-Var*, thus $(Q_0)\ \forall\,(Q_1)\ \tau_0 \sqsubseteq \sigma_0$ (6) holds by (4).

∘ Completeness For simplicity, we assume that $\mathsf{dom}(Q/\mathsf{ftv}(\sigma)) = \mathsf{dom}(Q)$ $\mathbf{(\underline{12})}$, that is, Q contains only useful bindings. By hypothesis, (5) and (6) hold, thus we have the premises (7), (8), and (9). By Rule Inst, (7) and (8), we have $(Q_0)\ \Gamma \vdash a : \forall\,(Q_1)\ \tau_1$ $\mathbf{(\underline{13})}$. By Lemma 3.4.4 applied to (9), there exists a prefix P_1 such that $\forall\,(P_1)\ \tau'_1 \to \tau'_0$ is an alpha-conversion of $\forall\,(Q_1)\ \tau_1 \to \tau_0$, and a substitution θ such that

$$(Q_0P_1)\ \theta(\alpha_1 \to \alpha_0) \equiv \tau'_1 \to \tau'_0\ (\underline{\mathbf{14}}) \qquad J = \mathsf{dom}(Q''/\alpha_1 \to \alpha_0)\ (\underline{\mathbf{15}}) \qquad \mathsf{dom}(\theta) \subseteq J$$

$$Q_0Q'' \sqsubseteq^{\mathsf{dom}(Q_0) \cup J} Q_0P_1\underline{\theta}\ (\underline{\mathbf{16}})$$

From (12) and (15), we get $J = \mathsf{dom}(Q'')$. Hence, (16) becomes $Q_0Q'' \sqsubseteq Q_0P_1\underline{\theta}$ $\mathbf{(\underline{17})}$. By Property 3.4.2.i (page 106) and PE-Free, we get $Q_0Q \sqsubseteq Q_0P_1\underline{\theta}$, that is (2) by taking $Q' = P_1\underline{\theta}$. By Property 1.5.11.viii (page 54) on (14), we get $(Q_0P_1)\ \theta(\alpha_1) \equiv \tau'_1$ $\mathbf{(\underline{18})}$ and $(Q_0P_1)\ \theta(\alpha_0) \equiv \tau'_0$ $\mathbf{(\underline{19})}$. By alpha-conversion, (13) becomes $(Q_0)\ \Gamma \vdash a : \forall\,(P_1)\ \tau'_1$. From (18), and Inst, we get $(Q_0)\ \Gamma \vdash a : \forall\,(P_1)\ \theta(\alpha_1)$, that is, (Q_0)

$\Gamma \vdash a : \forall\,(Q')\ \alpha_1$ (1) by taking $\sigma'_1 = \alpha_1$. We have $(Q_0 Q'')\ \sigma \mathrel{-\mkern-12mu\sqsubset} \alpha_1$ by A-Hyp. Hence, by Lemma 3.6.4 on (17), we get $(Q_0 Q')\ \sigma \mathrel{-\mkern-12mu\sqsubset} \alpha_1$, that is, (3). Similarly, we can derive $(Q_0 Q')\ \sigma \mathrel{-\mkern-12mu\sqsubset} \alpha_0$. Hence, $(Q_0)\ \forall\,(Q')\ \sigma \sqsubseteq \forall\,(Q')\ \alpha_0$ $(\underline{\mathbf{20}})$ holds by R-Context-R. We have $(Q_0)\ \forall\,(Q')\ \alpha_0 \equiv \forall\,(P_1)\ \tau'_0$ from (19). By alpha-conversion, we have $(Q_0)\ \forall\,(Q')\ \alpha_0 \equiv \forall\,(Q_1)\ \tau_0$ $(\underline{\mathbf{21}})$. By R-Trans on (20), (21), and (6) we get $(Q_0)\ \forall\,(Q')\ \sigma \sqsubseteq \sigma_0$, that is, (4). ■

Proof of Corollary 8.3.3

Directly, we assume $(Q_0)\ \Gamma \vdash (a : \star) : \sigma_0$ holds. By Lemma 6.2.6, $(Q_0)\ \Gamma \overset{\triangledown}{\vdash} (a : \star) : \sigma'_0$ $(\underline{\mathbf{1}})$ holds with $(Q_0)\ \sigma'_0 \sqsubseteq \sigma_0$ $(\underline{\mathbf{2}})$. The judgment (1) must be derived by Rule Oracle. Thus there must exists σ_1 and σ_2 such that we have

$$(Q_0)\ \Gamma \overset{\triangledown}{\vdash} a : \sigma_1\ (\underline{\mathbf{3}}) \qquad (Q_0)\ \sigma_1 \sqsubseteq \sigma_2\ (\underline{\mathbf{4}}) \qquad (Q_0)\ \sigma_2 \exists\ \sigma'_0\ (\underline{\mathbf{5}})$$

Let ϕ be a renaming of domain $\mathsf{dom}(Q_0)$. Let Q be $\phi(Q_0)$ and σ be $\phi(\sigma_0)$. Let Q' be $\underline{\phi^{\neg}}$. Let σ'_1 be $\phi(\sigma_2)$ and σ be $\phi(\sigma'_0)$. By Rule Inst, (3) and (4), we get $(Q_0)\ \Gamma \vdash a : \sigma_2$, which gives by Eq-Mono* $(Q_0)\ \Gamma \vdash a : \forall\,(Q')\ \sigma'_1$ $(\underline{\mathbf{6}})$. We have $Q_0 Q \sqsubseteq Q_0 Q'$ $(\underline{\mathbf{7}})$ by Property 3.4.2.v (page 106). From (5), we get $(Q_0 Q')\ \phi(\sigma'_0) \mathrel{-\mkern-12mu\sqsubset} \phi(\sigma_2)$, that is, $(Q_0 Q')\ \sigma \mathrel{-\mkern-12mu\sqsubset} \sigma'_1$ $(\underline{\mathbf{8}})$. By (2), we have $(Q_0)\ \forall\,(Q')\ \sigma \sqsubseteq \sigma_0$ $(\underline{\mathbf{9}})$. Hence, by (6), (7), (8), (9), and by Lemma 8.3.2 we have $(Q_0)\ \Gamma \vdash (a : \exists\,(Q)\ \sigma) : \sigma_0$. This is the expected result.

Conversely, by Lemma 8.3.2, there exists Q' and σ'_1 such that $(Q_0)\ \Gamma \vdash a : \forall\,(Q')\ \sigma'_1$ $(\underline{\mathbf{10}})$ holds as well as $(Q_0 Q')\ \sigma'_1 \exists\ \sigma$ $(\underline{\mathbf{11}})$ and $(Q_0)\ \forall\,(Q')\ \sigma \sqsubseteq \sigma_0$ $(\underline{\mathbf{12}})$. By Rule R-Context-R on (11), we get $(Q_0)\ \forall\,(Q')\ \sigma'_1 \exists\ \forall\,(Q')\ \sigma$ $(\underline{\mathbf{13}})$. Hence, by (10), (13) and Rule Oracle, we get $(Q_0)\ \Gamma \vdash (a : \star) : \forall\,(Q')\ \sigma$. We get the expected result by Rule Inst and (12). ■

Proof of Lemma 8.3.4

Hypothesis H0 is easily checked. Hypothesis H1 is a direct consequence of Corollary 8.3.3. Hypothesis H2: $(v : \exists\,(Q)\ \sigma)$ always reduces. ■

Proof of Lemma 10.2.1

By Lemmas 2.5.5 and 2.5.6, we have a restricted derivation of $(Q)\ \sigma_1 \mathrel{-\mkern-12mu\sqsubset}^{\bar{\alpha}} \sigma_2$ $(\underline{\mathbf{1}})$. We show $w_Y(\sigma_2) \leqslant_Z w_Y(\sigma_1)$ by induction on the derivation of (1). Case A-Equiv' is a direct consequence of Lemma 2.7.5. Case R-Trans is by induction hypothesis.

◦ Case A-Context-L': We have $\sigma_1 = \forall\,(\alpha = \sigma'_1)\ \sigma$, $\sigma_2 = \forall\,(\alpha = \sigma'_2)\ \sigma$, and the premise is $(Q)\ \sigma'_1 \mathrel{-\mkern-12mu\sqsubset} \sigma'_2$. By induction hypothesis, we have $w_Y(\sigma'_2) \leqslant_Z w_Y(\sigma'_1)$ $(\underline{\mathbf{2}})$. By Lemma 2.5.6, we have $\mathsf{nf}(\sigma) \neq \alpha$, and $\alpha \in \mathsf{ftv}(\sigma)$. Hence, by definition we have $w_Y(\sigma_1) = Y \times w_Y(\sigma'_1) + w_Y(\sigma)$ and $w_Y(\sigma_2) = Y \times w_Y(\sigma'_2) + w_Y(\sigma)$. We conclude by (2), then.

∘ CASE R-CONTEXT-R: We have $\sigma_1 = \forall\,(\alpha \diamond \sigma)\ \sigma'_1$, $\sigma_2 = \forall\,(\alpha \diamond \sigma)\ \sigma'_2$, and the premise is $(Q, \alpha \diamond \sigma)\ \sigma'_1 \sqsubseteq^{\bar{\alpha} \cup \{\alpha\}} \sigma'_2$ $\mathbf{(\underline{3})}$. By induction hypothesis, we have $w_Y(\sigma'_2) \leqslant_Z w_Y(\sigma'_1)$ $\mathbf{(\underline{4})}$. We consider a four subcases:

SUBCASE $\sigma \in \mathcal{T}$: Then $w_Y(\sigma_1) = w_Y(\sigma'_1)$ and $w_Y(\sigma_2) = w_Y(\sigma'_2)$. We conclude by (4). In the following, we assume $\sigma \notin \mathcal{T}$. By (3), and Lemmas 2.5.7 (page 83) and 2.1.4 (page 67), we have $\alpha \in \mathsf{ftv}(\sigma'_1)$ if and only if $\alpha \in \mathsf{ftv}(\sigma'_2)$ $\mathbf{(\underline{5})}$.

SUBCASE $\mathsf{nf}(\sigma'_1) = \alpha$: Then we have $(Q, \alpha \diamond \sigma)\ \sigma'_1 \equiv \sigma'_2$ by Lemma 2.1.6, which implies $(Q)\ \sigma_1 \equiv \sigma_2$. We conclude by Lemma 2.7.5.

SUBCASE $\alpha \notin \mathsf{ftv}(\sigma'_1)$: Then $\alpha \notin \mathsf{ftv}(\sigma'_2)$ by (5). Hence, we have $w_Y(\sigma_1) = w_Y(\sigma'_1)$ and $w_Y(\sigma_2) = w_Y(\sigma'_2)$. We conclude by (4).

SUBCASE $\alpha \in \mathsf{ftv}(\sigma'_1)$: Then $\alpha \in \mathsf{ftv}(\sigma'_2)$ by (5). Either $w_Y(\sigma_2)$ is $Y \times w_Y(\sigma) + w_Y(\sigma'_2)$ or it is $w_Y(\sigma)$ if $\mathsf{nf}(\sigma'_2) = \alpha$. In both cases, we see that $w_Y(\sigma_2) =_Z w_Y(\sigma) + w_Y(\sigma'_2)$. By (4), we have $w_Y(\sigma_2) \leqslant_Z w_Y(\sigma) + w_Y(\sigma'_1)$, that is, $w_Y(\sigma_2) \leqslant_Z w_Y(\sigma_1)$.

∘ CASE A-HYP': We have $\sigma_2 = \alpha$, with $(\alpha = \sigma_1) \in Q$. We simply note that $w_Y(\sigma_2)$ is $w_Y(\alpha)$, that is, 0.

∘ CASE A-ALIAS': We check that $w_Y(\sigma_1) =_Z w_Y(\sigma_2)$.

∘ CASE A-UP': We check that $w_Y(\sigma_1) =_Z w_Y(\sigma_2)$. ■

Proof of Property 10.3.3

Property i: Let S be an $\bar{\alpha}$-stable set. Let θ be a substitution invariant on $\bar{\alpha}$. By definition, we have $\theta(S) \subseteq S$. This implies $\mathsf{fsub}\,(\theta(S)) \subseteq \mathsf{fsub}\,(S)$ $\mathbf{(\underline{1})}$. By Property 10.3.1.ii, we have $\theta(\mathsf{fsub}\,(S)) \subseteq \mathsf{fsub}\,(\theta(S))$ $\mathbf{(\underline{2})}$. By (2) and (1), we get $\theta(\mathsf{fsub}\,(S)) \subseteq \mathsf{fsub}\,(S)$. This means that $\mathsf{fsub}\,(S)$ is $\bar{\alpha}$-stable.

Property ii: This is a direct consequence of Property i, by observing that $\{t\}$ is $\mathsf{ftv}(t)$-stable.

Property iii: By hypothesis, S is $\bar{\alpha}$-stable $\mathbf{(\underline{1})}$ and $\bar{\beta}$-stable $\mathbf{(\underline{2})}$. Let θ be a substitution invariant on $\bar{\alpha} \cap \bar{\beta}$ $\mathbf{(\underline{3})}$. We show by induction on the size of $\mathsf{dom}(\theta)$ that $\theta(S) \subseteq S$ holds. If $\mathsf{dom}(\theta) = \emptyset$, then result is immediate. Otherwise, θ can be decomposed into $[t/\alpha] \circ \theta'$, such that the size of $\mathsf{dom}(\theta')$ is the size of $\mathsf{dom}(\theta)$ minus one. We have to show $[t/\alpha] \circ \theta'(S) \subseteq S$. By induction hypothesis, we have $\theta'(S) \subseteq S$. Hence, it suffices to show that $S[t/\alpha] \subseteq S$ $\mathbf{(\underline{4})}$. If $\alpha \notin \bar{\alpha}$, the substitution $[t/\alpha]$ is invariant on $\bar{\alpha}$, thus (4) holds by (1). Otherwise, $\alpha \in \bar{\alpha}$. By hypothesis (3), we must have $\alpha \notin \bar{\alpha} \cap \bar{\beta}$. Hence, we have $\alpha \notin \bar{\beta}$. Then the substitution $[t/\alpha]$ is invariant on $\bar{\beta}$. Thus (4) holds by (2). In both cases, we have shown (4). Hence, by induction, we have $\theta(S) \subseteq S$. This holds for any θ invariant on $\bar{\alpha} \cap \bar{\beta}$, thus S is $(\bar{\alpha} \cap \bar{\beta})$-stable.

Property iv: By hypothesis, S_1 is $\bar{\alpha}$-stable $\mathbf{(\underline{1})}$ and S_2 is $\bar{\alpha}$-stable $\mathbf{(\underline{2})}$. Let t_1 be in S_1. If $t_1 \in \Sigma_{\bar{\beta}}^{\neg}$, then $t_1 \in S_1 \cap \Sigma_{\bar{\beta}}^{\neg}$, thus $t_1 \in S_2 \cap \Sigma_{\bar{\beta}}^{\neg}$ by hypothesis. Hence, $t_1 \in S_2$. Otherwise,

$t_1 \notin \Sigma^{\neg}_{\bar\beta}$, which means that some free variables of t_1 are in $\bar\beta$. Let ϕ be a renaming of domain $\bar\beta$ mapping $\bar\beta$ to fresh variables, that is, variables outside $\bar\alpha \cup \bar\beta \cup \mathsf{ftv}(t_1)$. Since we have $\bar\beta \# \bar\alpha$ by hypothesis, ϕ is invariant on $\bar\alpha$. Hence, $\phi(S_1) \subseteq S_1$ holds by (1). This implies $\phi(t_1) \in S_1$. Besides, $\phi(t_1) \in \Sigma^{\neg}_{\bar\beta}$. Hence, as shown above, $\phi(t_1) \in S_2$. We note that $\phi^{\neg}$ is a renaming invariant on $\bar\alpha \cup \bar\beta \cup \mathsf{ftv}(t_1)$. Hence, $\phi^{\neg}(S_2) \subseteq S_2$ holds by (2). This implies $\phi^{\neg} \circ \phi(t_1) \in S_2$, that is, $t_1 \in S_2$. As a conclusion, we have shown $S_1 \subseteq S_2$.

Property v: It is a direct consequence of Property iv.

Property vi: As a preliminary result, we show $\bigcup\limits_{\alpha \in I} \Sigma^{\neg}_{\bar\alpha \cup \{\alpha\}} = \Sigma^{\neg}_{\bar\alpha}$ $(\underline{\mathbf{1}})$. Indeed, we have $\Sigma^{\neg}_{\bar\alpha \cup \{\alpha\}} \subseteq \Sigma^{\neg}_{\bar\alpha}$ for any α. Hence, $\bigcup_{\alpha \in I} \Sigma^{\neg}_{\bar\alpha \cup \{\alpha\}} \subseteq \Sigma^{\neg}_{\bar\alpha}$ holds. Conversely, let t be in $\Sigma^{\neg}_{\bar\alpha}$. The set $\mathsf{ftv}(t)$ is finite and the set I is infinite. Hence, there exists $\alpha \in I$ such that $\alpha \notin \mathsf{ftv}(t)$. Then we have $t \in \Sigma^{\neg}_{\bar\alpha \cup \{\alpha\}}$, that is, $t \in \bigcup_{\alpha \in I} \Sigma^{\neg}_{\bar\alpha \cup \{\alpha\}}$. As a conclusion, we have shown (1).

We get back to the proof of the property. We have $S_1 \cap \Sigma^{\neg}_{\bar\alpha \cup \{\alpha\}} \subseteq S_2 \cap \Sigma^{\neg}_{\bar\alpha \cup \{\alpha\}}$ for all α in I. Hence, taking the union we get $\bigcup\limits_{\alpha \in I} S_1 \cap \Sigma^{\neg}_{\bar\alpha \cup \{\alpha\}} \subseteq \bigcup\limits_{\alpha \in I} S_2 \cap \Sigma^{\neg}_{\bar\alpha \cup \{\alpha\}}$, that is, $S_1 \cap \left(\bigcup\limits_{\alpha \in I} \Sigma^{\neg}_{\bar\alpha \cup \{\alpha\}} \right) \subseteq S_2 \cap \left(\bigcup\limits_{\alpha \in I} \Sigma^{\neg}_{\bar\alpha \cup \{\alpha\}} \right)$. By (1), we get $S_1 \cap \Sigma^{\neg}_{\bar\alpha} \subseteq S_2 \cap \Sigma^{\neg}_{\bar\alpha}$, which means $S_1 \subseteq_{\bar\alpha} S_2$.

Property vii: Direct consequence of Property vi.

Property viii: By hypothesis, we have $S_1 \subseteq_{\bar\alpha} S_2$ $(\underline{\mathbf{1}})$. We have to show $\mathsf{fsub}\,(S_1) \subseteq_{\bar\alpha} \mathsf{fsub}\,(S_2)$. Let t be in $\mathsf{fsub}\,(S_1) \cap \Sigma^{\neg}_{\bar\alpha}$. By definition, we have $\mathsf{ftv}(t) \# \bar\alpha$, and there exists t_1 in S_1 such that $t_1 \sqsubseteq_F t$ $(\underline{\mathbf{2}})$. Observing that $\mathsf{ftv}(t_1) \subseteq \mathsf{ftv}(t)$, we have $\mathsf{ftv}(t_1) \# \bar\alpha$. Hence, $t_1 \in S_1 \cap \Sigma^{\neg}_{\bar\alpha}$. From (1), we get $t_1 \in S_2$. Hence, from (2), we have $t \in \mathsf{fsub}\,(S_2)$. This is the expected result.

Property ix: Direct consequence of Property viii.

Property x: If we have $\bar\alpha \# \mathsf{codom}(\theta)$ and $S_1 =_{\bar\alpha} S_2$, then $\theta(S_1) =_{\bar\alpha} \theta(S_2)$. Let t be in $\theta(S_1) \cap \Sigma^{\neg}_{\bar\alpha}$. There exists t_1 in S_1 such that $t = \theta(t_1)$. Since $\bar\alpha \# \mathsf{ftv}(t)$, we must have $\bar\alpha \# \mathsf{ftv}(t_1)$. Hence, t_1 is in $S_1 \cap \Sigma^{\neg}_{\bar\alpha}$, which implies $t_1 \in S_2$. Then $\theta(t_1) \in \theta(S_2)$. We have shown $\theta(S_1) \subseteq_{\bar\alpha} \theta(S_2)$. By symmetry, we have $\theta(S_2) \subseteq_{\bar\alpha} \theta(S_1)$, thus $\theta(S_1) =_{\bar\alpha} \theta(S_2)$ holds.

■

Proof of Property 10.3.6

Property i: In all cases $(\!(\sigma)\!)$ is of the form $\mathsf{fsub}\,(S)$ for some set of types S, and we simply note that $\mathsf{fsub}\,(\mathsf{fsub}\,(S)) = \mathsf{fsub}\,(S)$.

Property ii: By structural induction on σ.

∘ Case σ is τ: Then $(\!(\sigma)\!)$ is $\mathsf{fsub}\,(\tau)$. Let t be in $(\!(\sigma)\!)$. We have $\tau \sqsubseteq_F t$, which implies that t is equivalent to τ. Hence, $\mathsf{ftv}(t) = \mathsf{ftv}(\tau)$, and $\forall\,\alpha \cdot t$ is equivalent to t. As a consequence, we have $\forall\,\alpha \cdot t \in (\!(\sigma)\!)$, which is the expected result.

∘ Case σ is $\bot$: Then $(\!(\sigma)\!)$ is $\mathsf{fsub}\,(\forall\,(\alpha)\;\alpha)$. We immediately have $\forall\,\alpha \cdot t \in (\!(\sigma)\!)$ for any t. Hence, $\forall\,\alpha \cdot (\!(\sigma)\!) \subseteq (\!(\sigma)\!)$.

∘ Case σ is $\forall\,(\beta = \sigma_1)\;\sigma_2$: We can freely assume $\beta \neq \alpha$. By definition, $(\!(\sigma)\!)$ is $\mathsf{fsub}\left((\!(\sigma_2)\!)[\underline{\sigma_1}/\beta]\right)$. Hence, $\forall\,\alpha \cdot (\!(\sigma)\!)$ is $\forall\,\alpha \cdot \mathsf{fsub}\left((\!(\sigma_2)\!)[\underline{\sigma_1}/\beta]\right)$. Thus, by Property 10.3.1.iii, we have $\forall\,\alpha \cdot (\!(\sigma)\!) \subseteq \mathsf{fsub}\left(\forall\,\alpha \cdot ((\!(\sigma_2)\!)[\underline{\sigma_1}/\beta])\right)$. By hypothesis, $\alpha \notin \mathsf{ftv}(\sigma)$ and σ is in normal form, hence $\alpha \notin \mathsf{ftv}(\sigma_1)$. By Property 10.3.1.i, we get $\alpha \notin \mathsf{ftv}(\underline{\sigma_1})$, thus $\forall\,\alpha\cdot$ and the substitution $[\underline{\sigma_1}/\beta]$ commute, leading to $\forall\,\alpha \cdot (\!(\sigma)\!) \subseteq \mathsf{fsub}\left((\forall\,\alpha \cdot (\!(\sigma_2)\!))[\underline{\sigma_1}/\beta]\right)$. By induction hypothesis, we have $\forall\,\alpha \cdot (\!(\sigma_2)\!) \subseteq (\!(\sigma_2)\!)$. Hence, $\forall\,\alpha \cdot (\!(\sigma)\!) \subseteq \mathsf{fsub}\left((\!(\sigma_2)\!)[\underline{\sigma_1}/\beta]\right)$ holds, that is, $\forall\,\alpha \cdot (\!(\sigma)\!) \subseteq (\!(\sigma)\!)$.

∘ Case σ is $\forall\,(\beta \geq \sigma_1)\;\sigma_2$: We can freely assume $\beta \neq \alpha$. By definition, $(\!(\sigma)\!)$ is

$$\mathsf{fsub}\left(\bigcup_{t_1 \in (\!(\sigma_1)\!)}^{\bar\alpha \# \mathsf{ftv}(\sigma_1,\sigma_2)} \forall\,\bar\alpha \cdot (\!(\sigma_2)\!)[t_1/\beta]\right)$$

Hence, we have

$$\begin{array}{rcll}
\forall\,\alpha \cdot (\!(\sigma)\!) & = & \forall\,\alpha \cdot \mathsf{fsub}\left(\bigcup_{t_1 \in (\!(\sigma_1)\!)}^{\bar\alpha \# \mathsf{ftv}(\sigma_1,\sigma_2)} \forall\,\bar\alpha \cdot (\!(\sigma_2)\!)[t_1/\beta]\right) & \\
 & \subseteq & \mathsf{fsub}\left(\bigcup_{t_1 \in (\!(\sigma_1)\!)}^{\bar\alpha \# \mathsf{ftv}(\sigma_1,\sigma_2)} \forall\,\alpha \cdot \forall\,\bar\alpha \cdot (\!(\sigma_2)\!)[t_1/\beta]\right) & \text{by Prop. 10.3.1.iii}
\end{array}$$

By hypothesis, $\alpha \notin \mathsf{ftv}(\sigma)$ and σ is in normal form. We simply note that $\alpha \cup \bar\alpha \;\#\; \mathsf{ftv}(\sigma_1, \sigma_2)$ holds for all $\bar\alpha$ disjoint from $\mathsf{ftv}(\sigma_1, \sigma_2)$. Hence, we have

$$\forall\,\alpha \cdot (\!(\sigma)\!) \subseteq \mathsf{fsub}\left(\bigcup_{t_1 \in (\!(\sigma_1)\!)}^{\bar\alpha \# \mathsf{ftv}(\sigma_1,\sigma_2)} \forall\,\bar\alpha \cdot (\!(\sigma_2)\!)[t_1/\beta]\right)$$

That is, $\forall\,\alpha \cdot (\!(\sigma)\!) \subseteq (\!(\sigma)\!)$.

Property iii: Let θ be a substitution invariant on $\mathsf{ftv}(\sigma)$. Let $\bar\alpha$ be its domain. By Property ii, we have $\forall\,\bar\alpha \cdot (\!(\sigma)\!) \subseteq (\!(\sigma)\!)$. Hence, applying $\mathsf{fsub}\,()$, we get $\mathsf{fsub}\,(\forall\,\bar\alpha \cdot (\!(\sigma)\!)) \subseteq \mathsf{fsub}\,((\!(\sigma)\!))$. By Property i, this is $\mathsf{fsub}\,(\forall\,\bar\alpha \cdot (\!(\sigma)\!)) \subseteq (\!(\sigma)\!)$. We simply note that

$\theta(\llparenthesis\sigma\rrparenthesis) \subseteq \mathsf{fsub}\,(\forall\,\bar\alpha\cdot\llparenthesis\sigma\rrparenthesis)$. Hence, we have $\theta(\llparenthesis\sigma\rrparenthesis) \subseteq \llparenthesis\sigma\rrparenthesis$. This holds for all θ invariant on $\mathsf{ftv}(\sigma)$, thus $\llparenthesis\sigma\rrparenthesis$ is $\mathsf{ftv}(\sigma)$-stable.

Property iv: Let t_1 be in $(\mathsf{fsub}\,(t))[t'/\alpha]$. By definition, there exists t_1' such that $t \sqsubseteq_F t_1'$ **(1)** and t_1 is $t_1'[t'/\alpha]$. We note that $t[t'/\alpha] \sqsubseteq_F t_1'[t'/\alpha]$ holds from (1), that is, $t[t'/\alpha] \sqsubseteq_F t_1$. Hence, t_1 is in $\mathsf{fsub}\,(t[t'/\alpha])$. We have shown $(\mathsf{fsub}\,(t))[t'/\alpha] \subseteq \mathsf{fsub}\,(t[t'/\alpha])$ **(2)**.

Conversely, let t_1 be in $\mathsf{fsub}\,(t[t'/\alpha]) \cap \Sigma_{\alpha}^{\neg}$. This means that $t[t'/\alpha] \sqsubseteq_F t_1$ **(3)** holds and $\alpha \notin \mathsf{ftv}(t_1)$ **(4)**. If $\alpha \notin \mathsf{ftv}(t)$, we have $t_1 \in \mathsf{fsub}\,(t)$, thus we have $t_1[t'/\alpha] \in (\mathsf{fsub}\,(t))[t'/\alpha]$, that is, by (4), $t_1 \in (\mathsf{fsub}\,(t))[t'/\alpha]$ **(5)**. From now on, we assume that $\alpha \in \mathsf{ftv}(t)$ **(6)**. By hypothesis and alpha-conversion, t is of the form $\forall\,\bar\alpha\cdot t_0$, with $\{\alpha\} \cup \mathsf{ftv}(t') \# \bar\alpha$ **(7)**, $t_0 \neq \alpha$, and t_0 is not a quantified type. Hence, $t[t'/\alpha]$ is equal to $\forall\,\bar\alpha\cdot(t_0[t'/\alpha])$, and $t_0[t'/\alpha]$ is not a quantified type. From (3), t_1 is of the form $\forall\,\bar\beta\cdot(t_0[t'/\alpha])[\bar t/\bar\alpha]$ **(8)**, for some types $\bar t$ and $\bar\beta$ disjoint from $\mathsf{ftv}(t_0[t'/\alpha])$ **(9)**. From (9) and (6), we get $\bar\beta \# \mathsf{ftv}(t')$ **(10)**. By alpha-conversion, we can freely assume that $\alpha \notin \bar\beta$ **(11)**. Hence, we have $\bar\beta \# \mathsf{ftv}(t_0)$ from (9) and (11). Therefore, $\forall\,\bar\alpha\cdot t_0 \sqsubseteq_F \forall\,\bar\beta\cdot t_0[\bar t/\bar\alpha]$ holds (by definition of $\sqsubseteq_F$). Let t_2 be $\forall\,\bar\beta\cdot t_0[\bar t/\bar\alpha]$. We have shown $t \sqsubseteq_F t_2$. Additionally, $\alpha \notin \mathsf{ftv}(\bar t)$ **(12)** holds from (4) and (8). We note that $t_1 = t_2[t'/\alpha]$ holds from (8), (11), (7), (12), and (10). Hence, we have shown that t_1 is in $(\mathsf{fsub}\,(t))[t'/\alpha]$. In both cases, we have shown (5). This implies $\mathsf{fsub}\,(t[t'/\alpha]) \cap \Sigma_{\alpha}^{\neg} \subseteq (\mathsf{fsub}\,(t))[t'/\alpha]$. Thus, $\mathsf{fsub}\,(t[t'/\alpha]) \subseteq_{\alpha} (\mathsf{fsub}\,(t))[t'/\alpha]$ **(13)** holds.

From (2) and (13), we have $\mathsf{fsub}\,(t[t'/\alpha]) =_{\alpha} (\mathsf{fsub}\,(t))[t'/\alpha]$.

Property v: By Property 10.3.1.ii, we have $\theta(\mathsf{fsub}\,(t)) \subseteq \mathsf{fsub}\,(\theta(t))$ **(1)**. Conversely, we show by induction on the size of $\mathsf{dom}(\theta)$ that $\mathsf{fsub}\,(\theta(t)) \subseteq_{\mathsf{dom}(\theta)} \theta(\mathsf{fsub}\,(t))$. If $\mathsf{dom}(\theta)$ is empty, the result is immediate. Otherwise, we can write θ in the form $\theta' \circ [t'/\alpha]$, with $\alpha \notin \mathsf{dom}(\theta')$ and the size of $\mathsf{dom}(\theta')$ is strictly smaller than the size of $\mathsf{dom}(\theta)$. By hypothesis, the normal form of t is not in $\mathsf{dom}(\theta)$. Hence, the normal form of t is not α. By Property iv, we get $\mathsf{fsub}\,(t[t'/\alpha]) =_{\alpha} (\mathsf{fsub}\,(t))[t'/\alpha]$ **(2)**. Let t_1 be in $\mathsf{fsub}\,(\theta(t)) \cap \Sigma_{\mathsf{dom}(\theta)}^{\neg}$ **(3)**. We have $t_1 \in \mathsf{fsub}\,(\theta_1(t[t'/\alpha]))$, and $t_1 \in \Sigma_{\mathsf{dom}(\theta_1)}^{\neg}$. By induction hypothesis, we get $t_1 \in \theta_1(\mathsf{fsub}\,(t[t'/\alpha]))$. Hence, there exists t_1' in $\mathsf{fsub}\,(t[t'/\alpha])$ such that $t_1 = \theta_1(t_1')$. We note that $\alpha \notin \mathsf{ftv}(t_1)$ holds from (3). It implies $\alpha \notin \mathsf{ftv}(t_1')$. Hence, t_1' is in $\mathsf{fsub}\,(t[t'/\alpha]) \cap \Sigma_{\alpha}^{\neg}$. By (2), we get $t_1' \in (\mathsf{fsub}\,(t))[t'/\alpha]$. Hence, t_1 is in $\theta_1(\mathsf{fsub}\,(t))[t'/\alpha]$, that is, $t_1 \in \theta(\mathsf{fsub}\,(t))$. In summary, we have shown $\mathsf{fsub}\,(\theta(t)) \subseteq_{\mathsf{dom}(\theta)} \theta(\mathsf{fsub}\,(t))$. With (1), we get $\mathsf{fsub}\,(\theta(t)) =_{\mathsf{dom}(\theta)} \theta(\mathsf{fsub}\,(t))$.

Property vi: First, we note that if $t_1 \sqsubseteq_F t_2$ holds, then we have $t_1 \leqslant_{/} t_2$ **(1)**. Then we proceed by structural induction on σ.

∘ Case σ is τ: Then $\llparenthesis\sigma\rrparenthesis$ is $\mathsf{fsub}\,(\tau)$, thus t is equivalent to τ, and $\tau/ = t/$ holds. Hence, we have $\tau \leqslant_{/} t$.

◦ Case σ is $\bot$: Then $\sigma \leqslant_{/} t$ always holds.

◦ Case σ is $\forall\,(\alpha_1 = \sigma_1)\ \sigma_2$: Then $(\!(\sigma)\!)$ is $\mathsf{fsub}\left((\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]\right)$. Thanks to Remark (1) above, it suffices to show the result for t in $(\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]$. Hence, we assume t of the form $t_2[\underline{\sigma_1}/\alpha]$ for $t_2 \in (\!(\sigma_2)\!)$. By induction hypothesis, we have $\sigma_2 \leqslant_{/} t_2$. Hence, we have $\mathsf{proj}(\sigma_2)[\mathsf{proj}(\sigma_1)/\alpha] \leqslant_{/} t_2[\mathsf{proj}(\sigma_1)/\alpha]$ by Property 2.1.2.ii. We note that $\mathsf{proj}(\sigma_2)[\mathsf{proj}(\sigma_1)/\alpha]$ is $\mathsf{proj}(\sigma)$, and that $\mathsf{proj}(\sigma_1) = \mathsf{proj}(\underline{\sigma_1})$. Hence, we get $\mathsf{proj}(\sigma) \leqslant_{/} t_2[\mathsf{proj}(\underline{\sigma_1})/\alpha]$, that is, $\sigma \leqslant_{/} t$, which is the expected result.

◦ Case σ is $\forall\,(\alpha_1 \geq \sigma_1)\ \sigma_2$: As above, it suffices to show the result for t in $\forall\,\bar\alpha \cdot (\!(\sigma_2)\!)[t_1/\alpha]$, with $t_1 \in (\!(\sigma_1)\!)$, and $\bar\alpha \,\#\, \mathsf{ftv}(\sigma_1, \sigma_2)$. Hence, t is of the form $\forall\,\bar\alpha \cdot t_2[t_1/\alpha]$, for $t_2 \in (\!(\sigma_2)\!)$. By induction hypothesis, we have $\mathsf{proj}(\sigma_2) \leqslant_{/} \mathsf{proj}(t_2)$. By Property 2.1.2.ii, we get $\mathsf{proj}(\sigma_2)[\mathsf{proj}(\sigma_1)/\alpha] \leqslant_{/} \mathsf{proj}(t_2)[\mathsf{proj}(\sigma_1)/\alpha]$ $\mathbf{(\underline{2})}$. By induction hypothesis, we have $\mathsf{proj}(\sigma_1) \leqslant_{/} \mathsf{proj}(t_1)$. Hence, by Property 2.1.2.iii, we get $\mathsf{proj}(t_2)[\mathsf{proj}(\sigma_1)/\alpha] \leqslant_{/} \mathsf{proj}(t_2)[\mathsf{proj}(t_1)/\alpha]$ $\mathbf{(\underline{3})}$. By transitivity of $\leqslant_{/}$, (2), and (3), we get $\mathsf{proj}(\sigma) \leqslant_{/} \mathsf{proj}(t_2[t_1/\alpha])$, that is, $\sigma \leqslant_{/} t_2[t_1/\alpha]$. It implies $\forall\,(\bar\alpha)\ \sigma \leqslant_{/} \forall\,\bar\alpha \cdot t_2[t_1/\alpha]$, that is $\forall\,(\bar\alpha)\ \sigma \leqslant_{/} t$. Observing that $\forall\,(\bar\alpha)\ \sigma \equiv \sigma$ by Eq-Free, we get $\sigma \leqslant_{/} t$ by Property 1.5.4.i.

Property vii: By hypothesis, $\mathsf{nf}(\sigma)$ is not $\bot$. By Property 2.1.5.iii, it implies $\sigma/\epsilon \neq \bot$. Hence, for all t' in $(\!(\sigma)\!)$, we have $t'/\epsilon = \sigma/\epsilon$ $\mathbf{(\underline{1})}$ by Property vi. By hypothesis, $\mathsf{nf}(\sigma)$ is not α, that is, $\sigma/\epsilon \neq \alpha$ by Property 2.1.5.ii. Hence, $t'/\epsilon \neq \alpha$ holds from (1), which implies that t' is not equivalent to α. As a consequence, $\mathsf{fsub}\,(t'[t/\alpha]) =_\alpha \mathsf{fsub}\,(t')\,[t/\alpha]$ holds by Property iv. This holds for all t' in $(\!(\sigma)\!)$, thus $\mathsf{fsub}\,((\!(\sigma)\!)[t/\alpha]) =_\alpha \mathsf{fsub}\,((\!(\sigma)\!))\,[t/\alpha]$. We conclude by Property i, then.

Property viii: We show by structural induction on σ that $(\!(\sigma)\!) = \mathsf{fsub}\,(\underline{\sigma})$.

◦ Case σ is τ: Then we have $(\!(\sigma)\!) = \mathsf{fsub}\,(\tau)$ and $\underline{\sigma} = \tau$, which gives $(\!(\sigma)\!) = \mathsf{fsub}\,(\underline{\sigma})$.

◦ Case σ is $\bot$: Then we have $(\!(\sigma)\!) = \mathsf{fsub}\,(\forall\,\alpha \cdot \alpha)$ and $\underline{\sigma} = \forall\,\alpha \cdot \alpha$. Hence, $(\!(\sigma)\!) = \mathsf{fsub}\,(\underline{\sigma})$.

◦ Case σ is $\forall\,(\alpha = \sigma_1)\ \sigma_2$: Then we have both $(\!(\sigma)\!) = \mathsf{fsub}\left((\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]\right)$ $\mathbf{(\underline{1})}$ and $\underline{\sigma} = \underline{\sigma_2}[\underline{\sigma_1}/\alpha]$. We note that $\underline{\sigma_2}$ is not α because σ is in normal form. Hence, we have $\mathsf{fsub}\,(\underline{\sigma}) =_\alpha \mathsf{fsub}\left(\underline{\sigma_2}\right)[\underline{\sigma_1}/\alpha]$ by Property iv. By induction hypothesis, we get $\mathsf{fsub}\,(\underline{\sigma}) =_\alpha (\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]$ $\mathbf{(\underline{2})}$. Since σ is in normal form, $\mathsf{nf}(\sigma_2)$ is neither α nor $\bot$, hence the relation $\mathsf{fsub}\left((\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]\right) =_\alpha (\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]$ $\mathbf{(\underline{3})}$ holds by Property vii. From (1), (2) and (3), we get $\mathsf{fsub}\,(\underline{\sigma}) =_\alpha (\!(\sigma)\!)$. This holds for any suitable choice of α, thus we get $\mathsf{fsub}\,(\underline{\sigma}) =_\emptyset (\!(\sigma)\!)$ by Property 10.3.3.vii. That is $\mathsf{fsub}\,(\underline{\sigma}) = (\!(\sigma)\!)$.

◦ Case σ is $\forall\,(\alpha \geq \bot)\ \sigma_2$: Then we have $(\!(\sigma)\!) = \mathsf{fsub}\left(\bigcup_{t \in (\!(\bot)\!)}^{\bar\alpha \# \mathsf{ftv}(\sigma_2)} \forall\,\bar\alpha \cdot (\!(\sigma_2)\!)[t/\alpha]\right)$ $\mathbf{(\underline{4})}$ and $\underline{\sigma} = \forall\,\alpha \cdot \underline{\sigma_2}$ $\mathbf{(\underline{5})}$. We note that $(\!(\bot)\!)$ is $\mathsf{fsub}\,(\forall\,\alpha \cdot \alpha)$ by definition. Hence, $t \in (\!(\bot)\!)$ holds for any t. Additionally, $\mathsf{fsub}\,(\underline{\sigma})$ is by definition the set of instances of $\underline{\sigma}$. From (5), it is the set $\bigcup_{t}^{\bar\alpha \# \mathsf{ftv}(\sigma_2)} \forall\,\bar\alpha \cdot \mathsf{fsub}\left(\underline{\sigma_2}\right)[t/\alpha]$. By induction hypothesis, we get

$\mathsf{fsub}\,(\underline{\sigma}) = \bigcup_{t}^{\bar\alpha \# \mathsf{ftv}(\sigma_2)} \forall\, \bar\alpha \cdot (\!(\sigma_2)\!)[t/\alpha]$. We see that $\mathsf{fsub}\,(\mathsf{fsub}\,(\underline{\sigma})) = (\!(\sigma)\!)$ holds from (4), that is, $\mathsf{fsub}\,(\underline{\sigma}) = (\!(\sigma)\!)$.

Property ix: We obviously have $\displaystyle\bigcup_{t \in S}^{\bar\alpha \# \bar\beta \cup \bar\gamma} \forall\, \bar\alpha \cdot S'[t/\alpha] \subseteq \bigcup_{t \in S}^{\bar\alpha \# \bar\beta} \forall\, \bar\alpha \cdot S'[t/\alpha]$. Hence, we only have to show $\displaystyle\bigcup_{t \in S}^{\bar\alpha \# \bar\beta} \forall\, \bar\alpha \cdot S'[t/\alpha] \subseteq \bigcup_{t \in S}^{\bar\alpha \# \bar\beta \cup \bar\gamma} \forall\, \bar\alpha \cdot S'[t/\alpha]$. Let t_1 be in the left-hand term. There exist $\bar\alpha$ disjoint from $\bar\beta$, t in S and t' in S' such that $t_1 = \forall\, \bar\alpha \cdot t'[t/\alpha]$. Let ϕ be a renaming of domain $\bar\alpha$ mapping its domain to fresh variables, that is, outside $\bar\alpha \cup \bar\beta \cup \bar\gamma \cup \mathsf{ftv}(t) \cup \mathsf{ftv}(t') \cup \{\alpha\}$. Let ϕ' be ϕ restricted to $\bar\alpha - \{\alpha\}$. We note that ϕ and ϕ' are invariant on $\bar\beta$. Hence, $\phi(S) \subseteq S$ and $\phi'(S') \subseteq S'$ hold. We note that $\forall\, \phi(\bar\alpha) \cdot \phi'(t')[\phi(t)/\alpha]$ is an alpha-conversion of t_1. Additionally, $\phi(\bar\alpha) \# \bar\beta \cup \bar\gamma$, $\phi'(t') \in S'$ and $\phi(t) \in S$. Hence, we have $t_1 \in \displaystyle\bigcup_{t \in S}^{\bar\alpha \# \bar\beta \cup \bar\gamma} \forall\, \bar\alpha \cdot S'[t/\alpha]$, which is the expected result.

Property x: Thanks to Property 10.3.1.ii (page 193), it suffices to show the relation $\mathsf{fsub}\,(\theta(S)) \subseteq_{\mathsf{dom}(\theta)} \theta(\mathsf{fsub}\,(S))$ **($\underline{1}$)**. Let t be in $\mathsf{fsub}\,(\theta(S)) \cap \Sigma_{\mathsf{dom}(\theta)}^{\neg}$ **($\underline{2}$)**. By definition, there exists t' in S such that $\theta(t') \sqsubseteq_F t$ **($\underline{3}$)**. We can freely assume that t' is in normal form. We consider two cases.

∘ Case $t' \in \mathsf{dom}(\theta)$: Then $\theta(t')$ is a monotype (by hypothesis), thus t and $\theta(t')$ are equivalent from (3). This implies that there exists t'' equivalent to t' such that $t = \theta(t'')$. Thus, we have $t \in \theta(\mathsf{fsub}\,(S))$.

∘ Otherwise we have $t' \notin \mathsf{dom}(\theta)$, and $t \notin \mathsf{dom}(\theta)$ holds from (2). From (3) we have $t \in \mathsf{fsub}\,(\theta(t'))$. By Property iv, we get $t \in \theta(\mathsf{fsub}\,(t'))$, which implies $t \in \theta(\mathsf{fsub}\,(S))$.

In both cases, we have $t \in \theta(\mathsf{fsub}\,(S))$. Hence, (1) is shown and $\mathsf{fsub}\,(\theta(S)) =_{\mathsf{dom}(\theta)} \theta(\mathsf{fsub}\,(S))$ holds.

Property xi: This is shown by structural induction on σ. Before all, we note that $\theta((\!(\sigma)\!)) \subseteq \Sigma_{\mathsf{dom}(\theta)}^{\neg}$ holds since θ is idempotent by convention. Hence, $\theta((\!(\sigma)\!)) =_{\mathsf{dom}(\theta)} (\!(\theta(\sigma))\!)$ implies $\theta((\!(\sigma)\!)) \subseteq (\!(\theta(\sigma))\!)$. We will implicitly apply this result to the induction hypothesis in some of the following cases.

∘ Case σ is τ: Then $(\!(\sigma)\!)$ is $\mathsf{fsub}\,(\tau)$, thus $\theta((\!(\sigma)\!))$ is $\mathsf{fsub}\,(\theta(\tau))$, that is, $(\!(\theta(\tau))\!)$.

∘ Case σ is $\bot$: Then $(\!(\theta(\sigma))\!)$ is $\mathsf{fsub}\,(\forall\, \alpha \cdot \alpha)$, that is, $(\!(\bot)\!)$. Hence, it remains to show $(\!(\bot)\!) =_{\mathsf{dom}(\theta)} \theta((\!(\bot)\!))$ **($\underline{1}$)**. We clearly have $\theta((\!(\bot)\!)) \subseteq (\!(\bot)\!)$ **($\underline{2}$)**. Conversely, let t be in $(\!(\bot)\!) \cap \Sigma_{\mathsf{dom}(\theta)}^{\neg}$. We have $\mathsf{ftv}(t) \# \mathsf{dom}(\theta)$. Hence, $\theta(t) = t$. Thus, t is in $\theta((\!(\bot)\!))$. We have shown $(\!(\bot)\!) \subseteq_{\mathsf{dom}(\theta)} \theta((\!(\bot)\!))$ **($\underline{3}$)**. From (2) and (3), we get (1).

∘ Case σ is $\forall\,(\alpha = \sigma_1)\;\sigma_2$: We can freely consider $\alpha \notin \mathsf{dom}(\theta) \cup \mathsf{codom}(\theta)$. We have $\llparenthesis\sigma\rrparenthesis = \mathsf{fsub}\left(\llparenthesis\sigma_2\rrparenthesis[\underline{\sigma_1}/\alpha]\right)$ **($\underline{\mathbf{4}}$)**. Besides, $\theta(\sigma)$ is $\forall\,(\alpha = \theta(\sigma_1))\;\theta(\sigma_2)$, thus $\llparenthesis\theta(\sigma)\rrparenthesis$ is by definition the set $\mathsf{fsub}\left(\llparenthesis\theta(\sigma_2)\rrparenthesis[\underline{\theta(\sigma_1)}/\alpha]\right)$ **($\underline{\mathbf{5}}$)**. We have

$$\begin{array}{llcll}
 & \theta(\llparenthesis\sigma\rrparenthesis) & = & \theta(\mathsf{fsub}\left(\llparenthesis\sigma_2\rrparenthesis[\underline{\sigma_1}/\alpha]\right)) & \text{from (4)} \\
 & & \subseteq & \mathsf{fsub}\left(\theta(\llparenthesis\sigma_2\rrparenthesis[\underline{\sigma_1}/\alpha])\right) & \text{by Property 10.3.1.ii} \\
 & & = & \mathsf{fsub}\left(\theta(\llparenthesis\sigma_2\rrparenthesis)[\theta(\underline{\sigma_1})/\alpha]\right) & \\
 & & = & \mathsf{fsub}\left(\theta(\llparenthesis\sigma_2\rrparenthesis)[\underline{\theta(\sigma_1)}/\alpha]\right) & \text{since } \theta(\underline{\sigma_1}) = \underline{\theta(\sigma_1)} \text{ (easy)} \\
(\underline{\mathbf{6}}) & \theta(\llparenthesis\sigma\rrparenthesis) & \subseteq & \mathsf{fsub}\left(\theta(\llparenthesis\sigma_2\rrparenthesis)[\underline{\theta(\sigma_1)}/\alpha]\right) & \text{as a summary}
\end{array}$$

Additionally, we have

$$\begin{array}{llcl}
(\underline{\mathbf{7}}) & \theta(\llparenthesis\sigma_2\rrparenthesis) & \subseteq & \llparenthesis\theta(\sigma_2)\rrparenthesis \\
(\underline{\mathbf{8}}) & \theta(\llparenthesis\sigma_2\rrparenthesis)[\underline{\theta(\sigma_1)}/\alpha] & \subseteq & \llparenthesis\theta(\sigma_2)\rrparenthesis[\underline{\theta(\sigma_1)}/\alpha] \\
(\underline{\mathbf{9}}) & \mathsf{fsub}\left(\theta(\llparenthesis\sigma_2\rrparenthesis)[\underline{\theta(\sigma_1)}/\alpha]\right) & \subseteq & \mathsf{fsub}\left(\llparenthesis\theta(\sigma_2)\rrparenthesis[\underline{\theta(\sigma_1)}/\alpha]\right)
\end{array}$$

We have (7) by induction hypothesis, (8) by applying $[\underline{\theta(\sigma_1)}/\alpha]$ to (7), and (9) by applying $\mathsf{fsub}\,()$ to (8). Combining (6), (9), and (5), we get $\theta(\llparenthesis\sigma\rrparenthesis) \subseteq \llparenthesis\theta(\sigma)\rrparenthesis$ **($\underline{\mathbf{10}}$)**.

Conversely, we show $\llparenthesis\theta(\sigma)\rrparenthesis \subseteq_{\mathsf{dom}(\theta)\cup\{\alpha\}} \theta(\llparenthesis\sigma\rrparenthesis)$. Since σ is in normal form, $\mathsf{nf}(\sigma_2)$ is neither α nor $\bot$. Hence, $\mathsf{nf}(\theta(\sigma_2))$ is neither α nor $\bot$. Then by Property vii and (5), we get $\llparenthesis\theta(\sigma)\rrparenthesis =_{\alpha} \llparenthesis\theta(\sigma_2)\rrparenthesis[\underline{\theta(\sigma_1)}/\alpha]$ **($\underline{\mathbf{11}}$)**. Let t be in $\llparenthesis\theta(\sigma)\rrparenthesis \cap \Sigma^{\neg}_{\mathsf{dom}(\theta)\cup\{\alpha\}}$ **($\underline{\mathbf{12}}$)**. From (11), we have $t \in \llparenthesis\theta(\sigma_2)\rrparenthesis[\underline{\theta(\sigma_1)}/\alpha]$. Hence, there exists $t_2 \in \llparenthesis\theta(\sigma_2)\rrparenthesis$ **($\underline{\mathbf{13}}$)** such that $t = t_2[\underline{\theta(\sigma_1)}/\alpha]$ **($\underline{\mathbf{14}}$)**. By induction hypothesis, we have $\llparenthesis\theta(\sigma_2)\rrparenthesis =_{\mathsf{dom}(\theta)} \theta(\llparenthesis\sigma_2\rrparenthesis)$ **($\underline{\mathbf{15}}$)**. Since we have $\mathsf{ftv}(t) \mathrel{\#} \mathsf{dom}(\theta)$ from (12), we get $\mathsf{ftv}(t_2) \mathrel{\#} \mathsf{dom}(\theta)$ **($\underline{\mathbf{16}}$)** by (14). From (13), (16), and (15), we have $t_2 \in \theta(\llparenthesis\sigma_2\rrparenthesis)$. Hence, t is in $\theta(\llparenthesis\sigma_2\rrparenthesis)[\underline{\theta(\sigma_1)}/\alpha]$, that is, in $\theta(\llparenthesis\sigma_2\rrparenthesis[\underline{\sigma_1}/\alpha])$. This implies $t \in \theta(\mathsf{fsub}\left(\llparenthesis\sigma_2\rrparenthesis[\underline{\sigma_1}/\alpha]\right))$. Hence, we have $t \in \theta(\llparenthesis\sigma\rrparenthesis)$. This holds for all t in $\llparenthesis\theta(\sigma)\rrparenthesis \cap \Sigma^{\neg}_{\mathsf{dom}(\theta)\cup\{\alpha\}}$, hence we have $\llparenthesis\theta(\sigma)\rrparenthesis \subseteq_{\mathsf{dom}(\theta)\cup\{\alpha\}} \theta(\llparenthesis\sigma\rrparenthesis)$ **($\underline{\mathbf{17}}$)**. By (10) and (17), we have $\llparenthesis\theta(\sigma)\rrparenthesis =_{\mathsf{dom}(\theta)\cup\{\alpha\}} \theta(\llparenthesis\sigma\rrparenthesis)$. This holds for any choice of α. Hence, by Property 10.3.3.vii, we get $\llparenthesis\theta(\sigma)\rrparenthesis =_{\mathsf{dom}(\theta)} \theta(\llparenthesis\sigma\rrparenthesis)$.

∘ Case σ is $\forall\,(\alpha \geq \sigma_1)\;\sigma_2$: We can freely assume $\alpha \mathrel{\#} \mathsf{dom}(\theta) \cup \mathsf{codom}(\theta)$. By definition, we have $\llparenthesis\sigma\rrparenthesis = \mathsf{fsub}\left(\bigcup_{t_1 \in \llparenthesis\sigma_1\rrparenthesis}^{\bar{\alpha} \# \mathsf{ftv}(\sigma_1,\sigma_2)} \forall\,\bar{\alpha}\cdot\llparenthesis\sigma_2\rrparenthesis[t_1/\alpha]\right)$. Hence, the set $\theta(\llparenthesis\sigma\rrparenthesis)$ is equal to $\bigcup_{t_1 \in \llparenthesis\sigma_1\rrparenthesis}^{\bar{\alpha} \# \mathsf{ftv}(\sigma_1,\sigma_2)} \theta(\mathsf{fsub}\,(\forall\,\bar{\alpha}\cdot\llparenthesis\sigma_2\rrparenthesis[t_1/\alpha]))$. Let J be $\mathsf{ftv}(\sigma_1) \cup \mathsf{ftv}(\sigma_2) \cup \{\alpha\} \cup \mathsf{dom}(\theta) \cup \mathsf{codom}(\theta)$. By Property iii, $\llparenthesis\sigma_1\rrparenthesis$ and $\llparenthesis\sigma_2\rrparenthesis$ are J-stable. By Property ix, we have $\theta(\llparenthesis\sigma\rrparenthesis) =$

$\bigcup_{t_1 \in \llparenthesis \sigma_1 \rrparenthesis}^{\bar\alpha \# J} \theta(\mathsf{fsub}\,(\forall\,\bar\alpha \cdot \llparenthesis \sigma_2 \rrparenthesis[t_1/\alpha]))$ (**18**). Additionally, $\theta(\sigma)$ is $\forall\,(\alpha = \theta(\sigma_1))\ \theta(\sigma_2)$, thus $\llparenthesis \theta(\sigma) \rrparenthesis$ is $\bigcup_{t_1 \in \llparenthesis \theta(\sigma_1) \rrparenthesis}^{\bar\alpha \# \mathsf{ftv}(\sigma_1, \sigma_2)} \mathsf{fsub}\,(\forall\,\bar\alpha \cdot \llparenthesis \theta(\sigma_2) \rrparenthesis[t_1/\alpha])$. By Property iii, $\llparenthesis \theta(\sigma_1) \rrparenthesis$ and $\llparenthesis \theta(\sigma_2) \rrparenthesis$ are J-stable. By Property ix, $\llparenthesis \theta(\sigma) \rrparenthesis = \bigcup_{t_1 \in \llparenthesis \theta(\sigma_1) \rrparenthesis}^{\bar\alpha \# J} \mathsf{fsub}\,(\forall\,\bar\alpha \cdot \llparenthesis \theta(\sigma_2) \rrparenthesis[t_1/\alpha])$ (**19**) holds. Let t_1 be in $\llparenthesis \sigma_1 \rrparenthesis$ and $\bar\alpha$ disjoint from J. We have the following:

$$
\begin{array}{lrcl}
(\mathbf{20}) & \theta(\mathsf{fsub}\,(\forall\,\bar\alpha \cdot \llparenthesis \sigma_2 \rrparenthesis[t_1/\alpha])) & \subseteq & \mathsf{fsub}\,(\theta(\forall\,\bar\alpha \cdot \llparenthesis \sigma_2 \rrparenthesis[t_1/\alpha])) \\
(\mathbf{21}) & & \subseteq & \mathsf{fsub}\,(\forall\,\bar\alpha \cdot \theta(\llparenthesis \sigma_2 \rrparenthesis[t_1/\alpha])) \\
(\mathbf{22}) & & = & \mathsf{fsub}\,(\forall\,\bar\alpha \cdot \theta(\llparenthesis \sigma_2 \rrparenthesis)[\theta(t_1)/\alpha]) \\
(\mathbf{23}) & \theta(\mathsf{fsub}\,(\forall\,\bar\alpha \cdot \llparenthesis \sigma_2 \rrparenthesis[t_1/\alpha])) & \subseteq & \mathsf{fsub}\,(\forall\,\bar\alpha \cdot \llparenthesis \theta(\sigma_2) \rrparenthesis[\theta(t_1)/\alpha]))
\end{array}
$$

We have (20) by Property 10.3.1.ii, (21) by commutation of θ and $\forall\,\bar\alpha\cdot$ since $\bar\alpha \# J$, (22) by propagating θ, and (23) since $\theta(\llparenthesis \sigma_2 \rrparenthesis) \subseteq \llparenthesis \theta(\sigma_2) \rrparenthesis$ holds by induction hypothesis. We have (23) for all t_1 in $\llparenthesis \sigma_1 \rrparenthesis$ and $\bar\alpha$ disjoint from J. Hence, taking the union, we get by (18)

$$\theta(\llparenthesis \sigma \rrparenthesis) \subseteq \bigcup_{t_1 \in \llparenthesis \sigma_1 \rrparenthesis}^{\bar\alpha \# J} \mathsf{fsub}\,(\forall\,\bar\alpha \cdot \llparenthesis \theta(\sigma_2) \rrparenthesis[\theta(t_1)/\alpha]))$$

Observing that the sets $\{\theta(t_1) \mid t_1 \in \llparenthesis \sigma_1 \rrparenthesis\}$ and $\{t_1 \mid t_1 \in \theta(\llparenthesis \sigma_1 \rrparenthesis)\}$ are equal, we get

$$\theta(\llparenthesis \sigma \rrparenthesis) \subseteq \bigcup_{t_1 \in \theta(\llparenthesis \sigma_1 \rrparenthesis)}^{\bar\alpha \# J} \mathsf{fsub}\,(\forall\,\bar\alpha \cdot \llparenthesis \theta(\sigma_2) \rrparenthesis[t_1/\alpha])) \quad (\mathbf{24})$$

By induction hypothesis, we have $\theta(\llparenthesis \sigma_1 \rrparenthesis) \subseteq \llparenthesis \theta(\sigma_1) \rrparenthesis$. Hence, by (24) and (19), we get $\theta(\llparenthesis \sigma \rrparenthesis) \subseteq \llparenthesis \theta(\sigma) \rrparenthesis$ **(25)**. Conversely, we show that $\llparenthesis \theta(\sigma) \rrparenthesis \subseteq_{\mathsf{dom}(\theta)\cup\{\alpha\}} \theta(\llparenthesis \sigma \rrparenthesis)$ holds. Let t be in $\llparenthesis \theta(\sigma) \rrparenthesis \cap \Sigma^{\neg}_{\mathsf{dom}(\theta)\cup\{\alpha\}}$ **(26)**. From (19), there exists $t_1 \in \llparenthesis \theta(\sigma_1) \rrparenthesis$ **(27)**, $\bar\alpha \# J$, and $t_2 \in \llparenthesis \theta(\sigma_2) \rrparenthesis$ such that $\forall\,\bar\alpha \cdot t_2[t_1/\alpha] \sqsubseteq_F t$ **(28)**. We have $\mathsf{ftv}(t) \# \mathsf{dom}(\theta)$ **(29)** from (26). Thus, $\mathsf{ftv}(t_2) \# \mathsf{dom}(\theta)$ holds from (28). Hence, by induction hypothesis, we have $t_2 \in \theta(\llparenthesis \sigma_2 \rrparenthesis)$. Hence, there exists $t'_2 \in \llparenthesis \sigma_2 \rrparenthesis$ such that $t_2 = \theta(t'_2)$. From (28), we get $t \in \mathsf{fsub}\,(\forall\,\bar\alpha \cdot \theta(t'_2)[t_1/\alpha])$. If $\alpha \notin \mathsf{ftv}(t_2)$, then (28) can as well be written $\forall\,\bar\alpha \cdot t_2[t'_1/\alpha] \sqsubseteq_F t$ for some t'_1 in $\theta(\llparenthesis \sigma_1 \rrparenthesis)$, and we have $t \in \mathsf{fsub}\,(\forall\,\bar\alpha \cdot \theta(t'_2)[t'_1/\alpha])$ **(30)**. Otherwise, we have $\alpha \in \mathsf{ftv}(t_2)$, thus (28) and (29) imply $\mathsf{ftv}(t_1) \# \mathsf{dom}(\theta)$. Hence, by induction hypothesis and (27), we get $t_1 \in \theta(\llparenthesis \sigma_1 \rrparenthesis)$, that is, there exists $t'_1 \in \theta(\llparenthesis \sigma_1 \rrparenthesis)$ (namely $t'_1 = t_1$), such that $t \in \mathsf{fsub}\,(\forall\,\bar\alpha \cdot \theta(t'_2)[t'_1/\alpha])$. We see that (30) holds in both cases for some $t'_1 \in \theta(\llparenthesis \sigma_1 \rrparenthesis)$. Hence, there exists t''_1 in $\llparenthesis \sigma_1 \rrparenthesis$ such that $t'_1 = \theta(t''_1)$ and $t \in \mathsf{fsub}\,(\forall\,\bar\alpha \cdot \theta(t'_2)[\theta(t''_1)/\alpha])$, that is, $t \in \mathsf{fsub}\,(\theta(\forall\,\bar\alpha \cdot t'_2[t''_1/\alpha]))$. From (26) and Property x, we get $t \in \theta(\mathsf{fsub}\,(\forall\,\bar{\alpha} \cdot t'_2[t''_1/\alpha]))$. As a consequence, we have $t \in \bigcup^{\bar{\alpha} \# J}_{t''_1 \in (\!(\sigma_1)\!)} \theta(\mathsf{fsub}\left(\forall\,\bar{\alpha} \cdot (\!(\sigma_2)\!)[t''_1/\alpha]\right)$, that is, $t \in \theta((\!(\sigma)\!))$. This holds for all t in $(\!(\theta(\sigma))\!) \cap \Sigma^{\neg}_{\mathsf{dom}(\theta) \cup \{\alpha\}}$, thus we have $(\!(\theta(\sigma))\!) \subseteq_{\mathsf{dom}(\theta) \cup \{\alpha\}} \theta((\!(\sigma)\!))$ $(\underline{\mathbf{31}})$. From (25) and (31), we get $(\!(\theta(\sigma))\!) =_{\mathsf{dom}(\theta) \cup \{\alpha\}} \theta((\!(\sigma)\!))$. This holds for any choice of α, thus Property 10.3.3.vii gives $(\!(\theta(\sigma))\!) =_{\mathsf{dom}(\theta)} \theta((\!(\sigma)\!))$.

Property xii: First, we show that $(\!(\sigma)\!) = (\!(\mathsf{nf}(\sigma))\!)$. Then we show that if $\sigma \approx \sigma'$ holds for σ and σ' in normal form, then $(\!(\sigma)\!) = (\!(\sigma')\!)$ holds. We conclude by Property 1.5.11.i, then.

First, we show by induction on the number of universal quantifiers of σ that $(\!(\sigma)\!) = (\!(\mathsf{nf}(\sigma))\!)$. We proceed by case analysis on the shape of σ.

- Case τ: We have $\mathsf{nf}(\tau) = \tau$.
- Case $\bot$: We have $\mathsf{nf}(\bot) = \bot$.
- Case $\forall\,(\alpha = \sigma_1)\,\sigma_2$: We consider five subcases:

Subcase $\mathsf{nf}(\sigma_2) = \alpha$: We have $\mathsf{nf}(\sigma) = \mathsf{nf}(\sigma_1)$ and by definition $(\!(\sigma)\!)$ is $\mathsf{fsub}\left((\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]\right)$. By induction hypothesis, $(\!(\sigma_2)\!) = \mathsf{fsub}\,(\alpha)$. Hence, $(\!(\sigma)\!)$ equals $\mathsf{fsub}\left(\underline{\sigma_1}\right)$, that is, $\mathsf{fsub}\left(\underline{\mathsf{nf}(\sigma_1)}\right)$ (by definition). By Property viii, we get $(\!(\sigma)\!) = (\!(\mathsf{nf}(\sigma_1))\!)$.

Subcase $\mathsf{nf}(\sigma_2) = \bot$: Then, $\mathsf{nf}(\sigma)$ is $\bot$ and by definition the interpretation $(\!(\sigma)\!)$ is $\mathsf{fsub}\left((\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]\right)$. By induction hypothesis, $(\!(\sigma_2)\!) = \mathsf{fsub}\,(\forall\,\beta \cdot \beta)$. Hence, $(\!(\sigma)\!) = \mathsf{fsub}\left((\mathsf{fsub}\,(\forall\,\beta \cdot \beta))[\underline{\sigma_1}/\alpha]\right)$. By Property 10.3.1.v, we get $(\!(\sigma)\!) = \mathsf{fsub}\left((\forall\,\beta \cdot \beta)[\underline{\sigma_1}/\alpha]\right)$, that is, $(\!(\sigma)\!) = \mathsf{fsub}\,(\forall\,\beta \cdot \beta)$. Hence, $(\!(\sigma)\!) = (\!(\bot)\!)$.

From now on, we assume that $\mathsf{nf}(\sigma_2)$ is neither $\bot$ nor α, so that $(\!(\sigma)\!)$ is by definition $\mathsf{fsub}\left((\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]\right)$. Then by Property vii, we have $(\!(\sigma)\!) =_{\alpha} (\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]$ $(\underline{\mathbf{1}})$.

Subcase $\alpha \notin \mathsf{ftv}(\sigma_2)$: Then $\mathsf{nf}(\sigma)$ is $\mathsf{nf}(\sigma_2)$. From (1), we have $(\!(\sigma)\!) =_{\alpha} (\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]$ $(\underline{\mathbf{2}})$. By Property iii, $(\!(\sigma_2)\!)$ is $\mathsf{ftv}(\sigma_2)$-stable. As a consequence, the inclusion $(\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha] \subseteq (\!(\sigma_2)\!)$ $(\underline{\mathbf{3}})$ holds. Conversely, if t is in $(\!(\sigma_2)\!) \cap \Sigma^{\neg}_{\alpha}$, we have $\alpha \notin \mathsf{ftv}(t)$, thus $t[\underline{\sigma_1}/\alpha] = t$, which implies $t \in (\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]$. As a consequence, $(\!(\sigma_2)\!) \cap \Sigma^{\neg}_{\alpha} \subseteq (\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]$. This is by definition $(\!(\sigma_2)\!) \subseteq_{\alpha} (\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]$ $(\underline{\mathbf{4}})$. By (3) and (4), we get $(\!(\sigma_2)\!) =_{\alpha} (\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]$ $(\underline{\mathbf{5}})$. By (2) and (5), we get $(\!(\sigma)\!) =_{\alpha} (\!(\sigma_2)\!)$ $(\underline{\mathbf{6}})$. By Property iii, $(\!(\sigma)\!)$ is $\mathsf{ftv}(\sigma)$-stable, that is, $\mathsf{ftv}(\sigma_2)$-stable. Additionally, we recall that $\alpha \notin \mathsf{ftv}(\sigma_2)$. Hence, by Property 10.3.3.v and (6), we get $(\!(\sigma)\!) = (\!(\sigma_2)\!)$. By induction hypothesis, we have $(\!(\sigma)\!) = (\!(\mathsf{nf}(\sigma_2))\!)$, that is, $(\!(\sigma)\!) = (\!(\mathsf{nf}(\sigma))\!)$.

Subcase $\sigma_1 \in \mathcal{T}$: Let τ be $\mathsf{nf}(\sigma_1)$. We have $\mathsf{nf}(\sigma) = \mathsf{nf}(\sigma_2)[\tau/\alpha]$ $(\underline{\mathbf{7}})$. From (1), we have $(\!(\sigma)\!) =_{\alpha} (\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]$. By induction hypothesis, we get the relation $(\!(\sigma)\!) =_{\alpha} (\!(\mathsf{nf}(\sigma_2))\!)[\tau/\alpha]$ $(\underline{\mathbf{8}})$. By Property xi, we have $(\!(\mathsf{nf}(\sigma_2))\!)[\tau/\alpha] =_{\alpha} (\!(\mathsf{nf}(\sigma_2)[\tau/\alpha])\!)$ $(\underline{\mathbf{9}})$. By (7), (8), and (9), we get $(\!(\sigma)\!) =_{\alpha} (\!(\mathsf{nf}(\sigma))\!)$. This holds for any choice of α, thus we get $(\!(\sigma)\!) = (\!(\mathsf{nf}(\sigma))\!)$ by Property 10.3.3.vii.

Otherwise, $\mathsf{nf}(\sigma)$ is $\forall\,(\alpha = \mathsf{nf}(\sigma_1))\ \mathsf{nf}(\sigma_2)$, and $(\!(\sigma)\!)$ is $\mathsf{fsub}\left((\!(\sigma_2)\!)[\underline{\sigma_1}/\alpha]\right)$, that is, $\mathsf{fsub}\left((\!(\sigma_2)\!)[\underline{\mathsf{nf}(\sigma_1)}/\alpha]\right)$. By definition, $(\!(\mathsf{nf}(\sigma))\!)$ is $\mathsf{fsub}\left((\!(\mathsf{nf}(\sigma_2))\!)[\underline{\mathsf{nf}(\sigma_1)}/\alpha]\right)$. By induction hypothesis, it is equal to $\mathsf{fsub}\left((\!(\sigma_2)\!)[\underline{\mathsf{nf}(\sigma_1)}/\alpha]\right)$, that is, $(\!(\sigma)\!)$.

- Case $\forall\,(\alpha \geq \sigma_1)\ \sigma_2$: By definition, $(\!(\sigma)\!)$ is $\mathsf{fsub}\left(\bigcup_{t \in (\!(\sigma_1)\!)}^{\bar\alpha \# \mathsf{ftv}(\sigma_1,\sigma_2)} \forall\,\bar\alpha \cdot (\!(\sigma_2)\!)[t/\alpha]\right)$ ($\underline{\mathbf{10}}$).

We consider five subcases:

Subcase $\mathsf{nf}(\sigma_2) = \alpha$: From (10) and induction hypothesis, the set $(\!(\sigma)\!)$ is equal to $\mathsf{fsub}\left(\bigcup_{t \in (\!(\mathsf{nf}(\sigma_1))\!)}^{\bar\alpha \# \mathsf{ftv}(\sigma_1,\sigma_2)} \forall\,\bar\alpha \cdot t\right)$. This can also be written

$$(\!(\sigma)\!) = \bigcup_{\bar\alpha \# \mathsf{ftv}(\sigma_1,\sigma_2)} \mathsf{fsub}\,(\forall\,\bar\alpha \cdot (\!(\mathsf{nf}(\sigma_1))\!))\ \ (\underline{\mathbf{11}}).$$

We immediately have $(\!(\mathsf{nf}(\sigma_1))\!) \subseteq (\!(\sigma)\!)$ ($\underline{\mathbf{12}}$), taking $\bar\alpha = \emptyset$. Conversely, we have $\forall\,\bar\alpha \cdot (\!(\sigma_1)\!) \subseteq (\!(\mathsf{nf}(\sigma_1))\!)$ by Property ii. Hence, $\mathsf{fsub}\,(\forall\,\bar\alpha \cdot (\!(\sigma_1)\!)) \subseteq \mathsf{fsub}\,((\!(\mathsf{nf}(\sigma_1))\!))$ by applying $\mathsf{fsub}\,()$, that is, $\mathsf{fsub}\,(\forall\,\bar\alpha \cdot (\!(\sigma_1)\!)) \subseteq (\!(\mathsf{nf}(\sigma_1))\!)$ by Property i. Taking the union over $\bar\alpha$, and by (11), we get $(\!(\sigma)\!) \subseteq (\!(\mathsf{nf}(\sigma_1))\!)$ ($\underline{\mathbf{13}}$). By (12) and (13), we get $(\!(\sigma)\!) = (\!(\mathsf{nf}(\sigma_1))\!)$, which is the expected result.

Subcase $\mathsf{nf}(\sigma_2) = \bot$: From (10) and induction hypothesis, we have $(\!(\sigma)\!)$ equal to $\mathsf{fsub}\left(\bigcup_{t \in (\!(\mathsf{nf}(\sigma_1))\!)}^{\bar\alpha \# \mathsf{ftv}(\sigma_1,\sigma_2)} \forall\,\bar\alpha \cdot (\!(\bot)\!)[t/\alpha]\right)$. Observing that $(\!(\bot)\!)$ is $\mathsf{fsub}\,(\forall\,(\beta)\ \beta)$, and taking $\bar\alpha = \emptyset$, we have $\mathsf{fsub}\,(\mathsf{fsub}\,(\forall\,\beta \cdot \beta)\,[t/\alpha]) \subseteq (\!(\sigma)\!)$ Hence, by Property 10.3.1.v, we have $\mathsf{fsub}\,((\forall\,\beta \cdot \beta)[t/\alpha]) \subseteq (\!(\sigma)\!)$, that is, $\mathsf{fsub}\,(\forall\,\beta \cdot \beta) \subseteq (\!(\sigma)\!)$, that is, $(\!(\bot)\!) \subseteq (\!(\sigma)\!)$. Conversely, $(\!(\sigma)\!) \subseteq (\!(\bot)\!)$ obviously holds, thus we have $(\!(\sigma)\!) = (\!(\bot)\!)$, which is the expected result.

Subcase $\alpha \notin \mathsf{ftv}(\mathsf{nf}(\sigma_2))$: From (10) and induction hypothesis, we have $(\!(\sigma)\!)$ equal to $\mathsf{fsub}\left(\bigcup_{t \in (\!(\sigma_1)\!)}^{\bar\alpha \# \mathsf{ftv}(\sigma_1,\sigma_2)} \forall\,\bar\alpha \cdot (\!(\mathsf{nf}(\sigma_2))\!)[t/\alpha]\right)$ ($\underline{\mathbf{14}}$). We have $\forall\,\alpha \cdot (\!(\mathsf{nf}(\sigma_2))\!) \subseteq (\!(\mathsf{nf}(\sigma_2))\!)$ by Property ii. Hence, $(\forall\,\alpha \cdot (\!(\mathsf{nf}(\sigma_2))\!))[t/\alpha] \subseteq (\!(\mathsf{nf}(\sigma_2))\!)[t/\alpha]$ holds, that is, $\forall\,\alpha \cdot (\!(\mathsf{nf}(\sigma_2))\!) \subseteq (\!(\mathsf{nf}(\sigma_2))\!)[t/\alpha]$. Applying fsub, we get the inclusion $\mathsf{fsub}\,(\forall\,\alpha \cdot (\!(\mathsf{nf}(\sigma_2))\!)) \subseteq \mathsf{fsub}\,((\!(\mathsf{nf}(\sigma_2))\!)[t/\alpha])$. Observing that $(\!(\mathsf{nf}(\sigma_2))\!) \subseteq \mathsf{fsub}\,(\forall\,\alpha \cdot (\!(\mathsf{nf}(\sigma_2))\!))$ holds, we get $(\!(\mathsf{nf}(\sigma_2))\!) \subseteq \mathsf{fsub}\,((\!(\mathsf{nf}(\sigma_2))\!)[t/\alpha])$. Hence, we have $(\!(\mathsf{nf}(\sigma_2))\!) \subseteq (\!(\sigma)\!)$, taking $\bar\alpha = \emptyset$ in (14). Conversely, $(\!(\mathsf{nf}(\sigma_2))\!)$ is $\mathsf{ftv}(\sigma_2)$-stable by Property iii. Note that $[t/\alpha]$ is invariant on $\mathsf{ftv}(\sigma_2)$. Hence, $(\!(\mathsf{nf}(\sigma_2))\!)[t/\alpha] \subseteq (\!(\mathsf{nf}(\sigma_2))\!)$. This gives $\forall\,\bar\alpha \cdot (\!(\mathsf{nf}(\sigma_2))\!)[t/\alpha] \subseteq (\!(\mathsf{nf}(\sigma_2))\!)$ by Property ii. Hence, $(\!(\sigma)\!) \subseteq (\!(\mathsf{nf}(\sigma_2))\!)$ holds by (14), taking the union over t and $\bar\alpha$. We have shown $(\!(\sigma)\!) = (\!(\mathsf{nf}(\sigma_2))\!)$, which is the expected result.

Subcase $\sigma_1 \in \mathcal{T}$: Let τ be $\mathsf{nf}(\sigma_1)$. From (10) and induction hypothesis, $(\!(\sigma)\!)$ is equal to $\mathsf{fsub}\left(\bigcup_{t\in(\!(\tau)\!)}^{\bar\alpha\#\mathsf{ftv}(\sigma_1,\sigma_2)} \forall\,\bar\alpha\cdot(\!(\mathsf{nf}(\sigma_2))\!)[t/\alpha]\right)$ **($\underline{15}$)**. We note that $t \in (\!(\tau)\!)$ implies t equivalent to τ. Hence, $(\!(\sigma)\!)$ is also equal to $\mathsf{fsub}\left(\bigcup^{\bar\alpha\#\mathsf{ftv}(\sigma_1,\sigma_2)} \forall\,\bar\alpha\cdot(\!(\mathsf{nf}(\sigma_2))\!)[\tau/\alpha]\right)$. Taking $\bar\alpha = \emptyset$, we have $(\!(\mathsf{nf}(\sigma_2))\!)[\tau/\alpha] \subseteq (\!(\sigma)\!)$. By Property xi, we get $(\!(\mathsf{nf}(\sigma_2)[\tau/\alpha])\!) \subseteq_\alpha (\!(\sigma)\!)$, that is $(\!(\mathsf{nf}(\sigma))\!) \subseteq_\alpha (\!(\sigma)\!)$ **($\underline{16}$)**. Conversely, by Property xi we have $(\!(\mathsf{nf}(\sigma_2))\!)[t/\alpha] \subseteq (\!(\mathsf{nf}(\sigma_2)[t/\alpha])\!)$, that is, $(\!(\mathsf{nf}(\sigma_2))\!)[t/\alpha] \subseteq (\!(\mathsf{nf}(\sigma))\!)$. Hence, $\forall\,\bar\alpha\cdot(\!(\mathsf{nf}(\sigma_2))\!)[t/\alpha] \subseteq \forall\,\bar\alpha\cdot(\!(\mathsf{nf}(\sigma))\!)$. By Property ii, we get $\forall\,\bar\alpha\cdot(\!(\mathsf{nf}(\sigma_2))\!)[t/\alpha] \subseteq (\!(\mathsf{nf}(\sigma))\!)$. Hence, the inclusion $\mathsf{fsub}\,(\forall\,\bar\alpha\cdot(\!(\mathsf{nf}(\sigma_2))\!)[t/\alpha] \subseteq (\!(\mathsf{nf}(\sigma))\!))$. This holds for all $\bar\alpha$ disjoint from $\mathsf{ftv}(\sigma_1,\sigma_2)$ and all t in $(\!(\tau)\!)$, hence we get $(\!(\sigma)\!) \subseteq (\!(\mathsf{nf}(\sigma))\!)$ **($\underline{17}$)** from (15). From (16) and (17), we get $(\!(\sigma)\!) =_\alpha (\!(\mathsf{nf}(\sigma))\!)$. This holds for any choice of α, hence we have $(\!(\sigma)\!) = (\!(\mathsf{nf}(\sigma))\!)$ by Property 10.3.3.vii.

Otherwise: By induction hypothesis applied to σ_1 and σ_2, and (10), the set $(\!(\sigma)\!)$ is equal to $\mathsf{fsub}\left(\bigcup_{t\in(\!(\mathsf{nf}(\sigma_1))\!)}^{\bar\alpha\#\mathsf{ftv}(\sigma_1,\sigma_2)} \forall\,\bar\alpha\cdot(\!(\mathsf{nf}(\sigma_2))\!)[t/\alpha]\right)$. Hence $(\!(\sigma)\!) = (\!(\forall\,(\alpha \geq \mathsf{nf}(\sigma_1))\ \mathsf{nf}(\sigma_2))\!)$, that is $(\!(\sigma)\!) = (\!(\mathsf{nf}(\sigma))\!)$.

Then we have to show that if we have $\sigma \approx \sigma'$ for σ and σ' in normal form, then $(\!(\sigma)\!) = (\!(\sigma')\!)$ holds. The proof is by induction on the derivation of $\sigma \approx \sigma'$. Three rules define the relation $\approx$. Two of them are context rules. These cases are shown directly by induction hypothesis. The third case is the commutation rule: by hypothesis we have $\alpha_1 \notin \mathsf{ftv}(\sigma_2)$ and $\alpha_2 \notin \mathsf{ftv}(\sigma_1)$, and

$$\forall\,(\alpha_1 \diamond_1 \sigma_1, \alpha_2 \diamond_2 \sigma_2)\ \sigma \approx \forall\,(\alpha_2 \diamond_2 \sigma_2, \alpha_1 \diamond_1 \sigma_1)\ \sigma$$

Let σ_a be the left-hand type, and σ_b be the right-hand type. The definitions to compute the sets $(\!(\sigma_a)\!)$ and $(\!(\sigma_b)\!)$ depend on the symbols $\diamond_1$ and $\diamond_2$. In order to factorize the four possible cases, we consider the following remark: Let σ' be a type and σ'' be an F-type. Let S be $\{\underline{\sigma'}\}$. Then we note that $\mathsf{fsub}\,((\!(\sigma)\!)[\underline{\sigma'}/\alpha])$ can be written $\mathsf{fsub}\left(\bigcup_{t\in S}^{\bar\alpha\#\mathsf{ftv}(\sigma,\sigma')} \forall\,\bar\alpha\cdot(\!(\sigma)\!)[t/\alpha]\right)$. Hence, taking $S_1 \triangleq (\!(\sigma_1)\!)$ if $\diamond_1$ is flexible and $S_1 \triangleq \{\underline{\sigma_1}\}$ is $\diamond_1$ is rigid, $(\!(\sigma_a)\!)$ is equal to

$$\mathsf{fsub}\left(\bigcup_{t_1\in S_1}^{\bar\alpha\#\mathsf{ftv}(\sigma_a)\cup\{\alpha_1\}} \forall\,\bar\alpha\cdot(\!(\forall\,(\alpha_2 \diamond_2 \sigma_2)\ \sigma)\!)[t_1/\alpha_1]\right)$$

Taking $S_2 \triangleq (\!(\sigma_2)\!)$ if $\diamond_2$ is flexible and $S_2 \triangleq \{\underline{\sigma_2}\}$ if $\diamond_2$ is rigid, $(\!(\sigma_a)\!)$ is then equal to

$$\mathsf{fsub}\left(\bigcup_{t_1 \in S_1}^{\bar\alpha \# \mathsf{ftv}(\sigma_a) \cup \{\alpha_1\}} \forall\,\bar\alpha \cdot \mathsf{fsub}\left(\bigcup_{t_2 \in S_2}^{\bar\beta \# \mathsf{ftv}(\sigma_2, \sigma)} \forall\,\bar\beta \cdot (\!(\sigma)\!)[t_2/\alpha_2]\right)[t_1/\alpha_1]\right)$$

Let J be $\mathsf{ftv}(\sigma_1) \cup \mathsf{ftv}(\sigma_2) \cup \mathsf{ftv}(\sigma)$. By Properties iii and ix, we get

$$\mathsf{fsub}\left(\bigcup_{t_1 \in S_1}^{\bar\alpha \# J} \forall\,\bar\alpha \cdot \mathsf{fsub}\left(\bigcup_{t_2 \in S_2}^{\bar\beta \# J} \forall\,\bar\beta \cdot (\!(\sigma)\!)[t_2/\alpha_2]\right)[t_1/\alpha_1]\right)$$

By notation, it is equal to

$$\mathsf{fsub}\left(\bigcup_{t_1 \in S_1}^{\bar\alpha \# J} \bigcup_{t_2 \in S_2}^{\bar\beta \# J} \forall\,\bar\alpha \cdot \mathsf{fsub}\left(\forall\,\bar\beta \cdot (\!(\sigma)\!)[t_2/\alpha_2]\right)[t_1/\alpha_1]\right)$$

By Property 10.3.1.iii, we get

$$(\!(\sigma_a)\!) \subseteq \mathsf{fsub}\left(\bigcup_{t_1 \in S_1}^{\bar\alpha \# J} \bigcup_{t_2 \in S_2}^{\bar\beta \# J} \mathsf{fsub}\left(\forall\,\bar\alpha \cdot \forall\,\bar\beta \cdot (\!(\sigma)\!)[t_2/\alpha_2]\right)[t_1/\alpha_1]\right)$$

By notation, and observing that $\mathsf{fsub}\,(\mathsf{fsub}\,(S)) = \mathsf{fsub}\,(S)$, we get

$$(\!(\sigma_a)\!) \subseteq \mathsf{fsub}\left(\bigcup_{t_1 \in S_1}^{\bar\alpha \# J} \bigcup_{t_2 \in S_2}^{\bar\beta \# J} \forall\,\bar\alpha \cdot \forall\,\bar\beta \cdot (\!(\sigma)\!)[t_2/\alpha_2][t_1/\alpha_1]\right) \quad (\underline{\mathbf{18}})$$

Let $\bar\alpha$ and $\bar\beta$ be disjoint from J. Let t_1 and t_2 be in S_1 and S_2, respectively. Let γ_1 and γ_2 be fresh variables, that is, not in $J \cup \mathsf{ftv}(t_1) \cup \mathsf{ftv}(t_2) \cup \bar\alpha \cup \bar\beta \cup \{\alpha_1\} \cup \{\alpha_2\}$. Let t_1' be $t_1[\gamma_2/\alpha_2]$ and t_2' be $t_2[\gamma_1/\alpha_1]$. We recall that S_1 is either $(\!(\sigma_1)\!)$ or $\{\underline{\sigma_1}\}$. In the first case, S_1 is $\mathsf{ftv}(\sigma_1)$-stable by Property iii. By hypothesis, we have $\alpha_2 \notin \mathsf{ftv}(\sigma_1)$. Hence, $S_1[\gamma_2/\alpha_2] \subseteq S_1$, thus $t_1' \in S_1$ **(<u>19</u>)**. In the second case, t_1 is $\underline{\sigma_1}$ and $\mathsf{ftv}(t_1) = \mathsf{ftv}(\sigma_1)$ by Property 10.3.1.i, hence $t_1' = t_1$. In both cases, (19) holds. Similarly, $t_2' \in S_2$ holds. Additionally, we have $\{\gamma_1, \gamma_2\} \# J$. Hence, we have $\{\gamma_1, \gamma_2\} \cup \bar\alpha \# J$. As a consequence, we have

$$\forall\,\gamma_1\gamma_2\bar\alpha\bar\beta \cdot (\!(\sigma)\!)[t_1'/\alpha_1][t_2'/\alpha_2] \in \bigcup_{t_1 \in S_1}^{\bar\alpha \# J} \bigcup_{t_2 \in S_2}^{\bar\beta \# J} \forall\,\bar\alpha \cdot \forall\,\bar\beta \cdot (\!(\sigma)\!)[t_1/\alpha_1][t_2/\alpha_2]$$

Applying $\mathsf{fsub}\,()$, we get

$$\mathsf{fsub}\left(\forall\,\gamma_1\gamma_2\bar\alpha\bar\beta \cdot (\!(\sigma)\!)[t_1'/\alpha_1][t_2'/\alpha_2]\right) \subseteq \mathsf{fsub}\left(\bigcup_{t_1 \in S_1}^{\bar\alpha \# J} \bigcup_{t_2 \in S_2}^{\bar\beta \# J} \forall\,\bar\alpha \cdot \forall\,\bar\beta \cdot (\!(\sigma)\!)[t_1/\alpha_1][t_2/\alpha_2]\right)$$

Observing that we have $\llparenthesis\sigma\rrparenthesis[t'_1/\alpha_1][t'_2/\alpha_2] = \llparenthesis\sigma\rrparenthesis[t'_2/\alpha_2][t'_1/\alpha_1]$, we get

$$\mathsf{fsub}\left(\forall\,\gamma_1\gamma_2\bar{\alpha}\bar{\beta}\cdot\llparenthesis\sigma\rrparenthesis[t'_2/\alpha_2][t'_1/\alpha_1]\right) \subseteq \mathsf{fsub}\left(\bigcup_{t_1\in S_1}^{\bar{\alpha}\#J}\bigcup_{t_2\in S_2}^{\bar{\beta}\#J}\forall\,\bar{\alpha}\cdot\forall\,\bar{\beta}\cdot\llparenthesis\sigma\rrparenthesis[t_1/\alpha_1][t_2/\alpha_2]\right)$$

Additionally, we note that the type $\forall\,\bar{\alpha}\bar{\beta}\cdot\llparenthesis\sigma\rrparenthesis[t_2/\alpha_2][t_1/\alpha_1]$ belongs to the set

$$\mathsf{fsub}\left(\forall\,\gamma_1\gamma_2\bar{\alpha}\bar{\beta}\cdot\llparenthesis\sigma\rrparenthesis[t'_2/\alpha_2][t'_1/\alpha_1]\right)$$

Thus, we get

$$\forall\,\bar{\alpha}\bar{\beta}\cdot\llparenthesis\sigma\rrparenthesis[t_2/\alpha_2][t_1/\alpha_1] \in \mathsf{fsub}\left(\bigcup_{t_1\in S_1}^{\bar{\alpha}\#J}\bigcup_{t_2\in S_2}^{\bar{\beta}\#J}\forall\,\bar{\alpha}\cdot\forall\,\bar{\beta}\cdot\llparenthesis\sigma\rrparenthesis[t_1/\alpha_1][t_2/\alpha_2]\right)$$

This holds for all t_1 and t_2 in S_1 and S_2, respectively, and for all $\bar{\alpha}$ and $\bar{\beta}$ disjoint from J. Hence, taking the union over t_1, t_2, $\bar{\alpha}$ and $\bar{\beta}$, we get

$$\begin{array}{rl} & \mathsf{fsub}\left(\bigcup_{t_1\in S_1}^{\bar{\alpha}\#J}\bigcup_{t_2\in S_2}^{\bar{\beta}\#J}\forall\,\bar{\alpha}\bar{\beta}\cdot\llparenthesis\sigma\rrparenthesis[t_2/\alpha_2][t_1/\alpha_1]\right)\\ \subseteq & \mathsf{fsub}\left(\bigcup_{t_1\in S_1}^{\bar{\alpha}\#J}\bigcup_{t_2\in S_2}^{\bar{\beta}\#J}\forall\,\bar{\alpha}\cdot\forall\,\bar{\beta}\cdot\llparenthesis\sigma\rrparenthesis[t_1/\alpha_1][t_2/\alpha_2]\right)\end{array}$$

Applying $\mathsf{fsub}\,()$, and by (18), we get

$$\llparenthesis\sigma_a\rrparenthesis \subseteq \mathsf{fsub}\left(\bigcup_{t_1\in S_1}^{\bar{\alpha}\#J}\bigcup_{t_2\in S_2}^{\bar{\beta}\#J}\forall\,\bar{\alpha}\cdot\forall\,\bar{\beta}\cdot\llparenthesis\sigma\rrparenthesis[t_1/\alpha_1][t_2/\alpha_2]\right)$$

By commutation of the union symbols, we get

$$\llparenthesis\sigma_a\rrparenthesis \subseteq \mathsf{fsub}\left(\bigcup_{t_2\in S_2}^{\bar{\beta}\#J}\bigcup_{t_1\in S_1}^{\bar{\alpha}\#J}\forall\,\bar{\beta}\cdot\forall\,\bar{\alpha}\cdot\llparenthesis\sigma\rrparenthesis[t_1/\alpha_1][t_2/\alpha_2]\right)$$

This can also be written

$$\llparenthesis\sigma_a\rrparenthesis \subseteq \mathsf{fsub}\left(\bigcup_{t_2\in S_2}^{\bar{\beta}\#J}\forall\,\bar{\beta}\cdot\bigcup_{t_1\in S_1}\llparenthesis\sigma\rrparenthesis[t_1/\alpha_1][t_2/\alpha_2]\right)$$

Observing that we have $\bigcup_{t_1 \in S_1} (\!(\sigma)\!)[t_1/\alpha_1] \subseteq (\!(\forall\,(\alpha_1 \diamond_1 \sigma_1)\ \sigma)\!)$, we get

$$(\!(\sigma_a)\!) \subseteq \mathsf{fsub}\left(\bigcup_{t_2 \in S_2}^{\bar\beta \# J} \forall\,\bar\beta \cdot (\!(\forall\,(\alpha_1 \diamond_1 \sigma_1)\ \sigma)\!)[t_2/\alpha_2]\right)$$

That is,

$$(\!(\sigma_a)\!) \subseteq (\!(\forall\,(\alpha_2 \diamond_2 \sigma_2)\ \forall\,(\alpha_1 \diamond_1 \sigma_1)\ \sigma)\!)$$

Hence, we have shown $(\!(\sigma_a)\!) \subseteq (\!(\sigma_b)\!)$. By symmetry, we immediately get $(\!(\sigma_b)\!) \subseteq (\!(\sigma_a)\!)$. Hence, $(\!(\sigma_a)\!) = (\!(\sigma_b)\!)$, which is the expected result. Finally, we have shown that $\sigma \approx \sigma'$ implies $(\!(\sigma)\!) = (\!(\sigma')\!)$.

Property xiii: We prove the property by structural induction on Q. Let θ be in $(\!(Q)\!)$. We proceed by case analysis on the shape of Q.

- ○ Case $\emptyset$: Immediate.
- ○ Case $(Q', \alpha \diamond \sigma)$ with $\sigma \notin \mathcal{T}$: Then θ is equal to $\theta' \circ [t/\alpha]$ for t being $\underline{\sigma}$ if $\diamond$ is rigid, or t being in $(\!(\sigma)\!)$ otherwise. By induction hypothesis, there exists θ'' such that $\theta' = \theta'' \circ \widehat{Q'}$. We get the expected result by observing that $\widehat{Q} = \widehat{Q'}$.
- ○ Case $(Q', \alpha = \sigma)$ with $\sigma \in \mathcal{T}$: Then θ is equal to $\theta' \circ [\underline{\sigma}/\alpha]$ for some θ' in $(\!(Q')\!)$. Let τ be $\mathsf{nf}(\sigma)$. By induction hypothesis, there exists θ'' such that $\theta' = \theta'' \circ \widehat{Q'}$. Hence $\theta = \theta'' \circ \widehat{Q'} \circ [\underline{\sigma}/\alpha]$. By definition, $\underline{\sigma}$ is $\underline{\tau}$, that is, τ. Hence, $\theta = \theta'' \circ \widehat{Q'} \circ [\tau/\alpha]$. We conclude by observing that $\widehat{Q}$ is $\widehat{Q'} \circ [\tau/\alpha]$.
- ○ Case $(Q', \alpha \geq \sigma)$ with $\sigma \in \mathcal{T}$: Then θ is equal to $\theta' \circ [t/\alpha]$ for some t in $(\!(\sigma)\!)$, and θ' in $(\!(Q')\!)$. Let τ be $\mathsf{nf}(\sigma)$. By Property xii, we have $t \in (\!(\tau)\!)$, that is, $t \in \mathsf{fsub}\,(\tau)$. As a consequence, t is equivalent to τ. Thus, $\mathsf{nf}(t)$ is τ. By induction hypothesis, there exists θ'' such that $\theta' = \theta'' \circ \widehat{Q'}$. Hence we have $\theta = \theta'' \circ \widehat{Q'} \circ [\tau/\alpha]$. We conclude by observing that $\widehat{Q}$ is $\widehat{Q'} \circ [\tau/\alpha]$. ■

Proof of Lemma 10.3.8

By induction on the derivation of $(Q)\ \sigma_1 \sqsubseteq \sigma_2$ **(1)**. Let θ be in $(\!(Q)\!)$. We proceed by case analysis on the last rule in the derivation of (1).

○ Case A-Equiv: By hypothesis, $(Q)\ \sigma_1 \equiv \sigma_2$ holds. By Corollary 1.5.10, we have $\widehat{Q}(\sigma_1) \equiv \widehat{Q}(\sigma_2)$. Hence, by Property 10.3.6.xii, we have $(\!(\widehat{Q}(\sigma_1))\!) = (\!(\widehat{Q}(\sigma_2))\!)$. By Property iv, we get $\widehat{Q}((\!(\sigma_1)\!)) =_{\mathsf{dom}(\widehat{Q})} \widehat{Q}((\!(\sigma_2)\!))$ **(2)**. Since $\widehat{Q}$ is idempotent, we have $\widehat{Q}((\!(\sigma_1)\!)) \cap \Sigma^{\neg}_{\mathsf{dom}(\widehat{Q})} = \widehat{Q}((\!(\sigma_1)\!))$ as well as $\widehat{Q}((\!(\sigma_2)\!)) \cap \Sigma^{\neg}_{\mathsf{dom}(\widehat{Q})} = \widehat{Q}((\!(\sigma_2)\!))$. Hence, from (2), we get $\widehat{Q}((\!(\sigma_1)\!)) = \widehat{Q}((\!(\sigma_2)\!))$ **(3)**. By Property 10.3.6.xiii, we have $\theta = \theta' \circ \widehat{Q}$.

Composing (3) by θ', we get $\theta((\!(\sigma_1)\!)) = \theta((\!(\sigma_2)\!))$. This leads to $\mathsf{fsub}\,(\theta((\!(\sigma_1)\!))) = \mathsf{fsub}\,(\theta((\!(\sigma_2)\!)))$.

∘ Case R-Trans: By induction hypothesis.

∘ Case R-Context-Rigid: The premise is $(Q)\ \sigma'_1 \sqsubseteq \sigma'_2$ **($\underline{4}$)**, σ_1 is $\forall\,(\alpha = \sigma'_1)\ \sigma$, and σ_2 is $\forall\,(\alpha = \sigma'_2)\ \sigma$. Since σ_1 and σ_2 are shallow, σ'_1 and σ'_2 are F-types. Hence, we have $(\!(\sigma'_1)\!) = \mathsf{fsub}\left(\underline{\sigma'_1}\right)$ and $(\!(\sigma'_2)\!) = \mathsf{fsub}\left(\underline{\sigma'_2}\right)$ by Property iii. By induction hypothesis and (4), it gives $\mathsf{fsub}\,(\theta((\!(\sigma'_1)\!))) = \mathsf{fsub}\,(\theta((\!(\sigma'_2)\!)))$, that is, $\mathsf{fsub}\left(\theta(\mathsf{fsub}\left(\underline{\sigma'_1}\right))\right) = \mathsf{fsub}\left(\theta(\mathsf{fsub}\left(\underline{\sigma'_2}\right))\right)$. By Property 10.3.1.v, we get $\mathsf{fsub}\left(\theta(\underline{\sigma'_1})\right) = \mathsf{fsub}\left(\theta(\underline{\sigma'_2})\right)$ **($\underline{5}$)**. In particular, we have $\theta(\underline{\sigma'_1}) \in \mathsf{fsub}\left(\theta(\underline{\sigma'_1})\right)$. Hence, by (5), we have $\theta(\underline{\sigma'_1}) \in \mathsf{fsub}\left(\theta(\underline{\sigma'_2})\right)$, which means that $\theta(\underline{\sigma'_2}) \sqsubseteq_F \theta(\underline{\sigma'_1})$ **($\underline{6}$)** holds. Similarly, we show that $\theta(\underline{\sigma'_1}) \sqsubseteq_F \theta(\underline{\sigma'_2})$ **($\underline{7}$)** holds. As a consequence of (6) and (7), $\theta(\underline{\sigma'_1})$ and $\theta(\underline{\sigma'_2})$ are equivalent **($\underline{8}$)** in System F. Now, by definition, $(\!(\sigma_1)\!)$ is $(\!(\sigma)\!)[\underline{\sigma'_1}/\alpha]$. Hence, $\theta((\!(\sigma_1)\!))$ is $\theta((\!(\sigma)\!))[\theta(\underline{\sigma'_1})/\alpha]$. By (8), this implies $\theta((\!(\sigma)\!))[\theta(\underline{\sigma'_2})/\alpha] \subset \mathsf{fsub}\,(\theta((\!(\sigma_1)\!)))$, that is, $\theta((\!(\sigma_2)\!)) \subset \mathsf{fsub}\,(\theta((\!(\sigma_1)\!)))$. Similarly, we have $\theta((\!(\sigma_1)\!)) \subset \mathsf{fsub}\,(\theta((\!(\sigma_2)\!)))$. As a result, $\theta((\!(\sigma_1)\!))$ and $\theta((\!(\sigma_2)\!))$ are equivalent in System F. This implies $\mathsf{fsub}\,(\theta((\!(\sigma_1)\!))) = \mathsf{fsub}\,(\theta((\!(\sigma_2)\!)))$.

∘ Case R-Context-R: The premise is $(Q, \alpha \diamond \sigma)\ \sigma'_1 \sqsubseteq \sigma'_2$, σ_1 is $\forall\,(\alpha \diamond \sigma)\ \sigma'_1$, and σ_2 is $\forall\,(\alpha \diamond \sigma)\ \sigma'_2$.

Subcase $\diamond$ is rigid: Then $(\!(\sigma_1)\!)$ is $\mathsf{fsub}\,((\!(\sigma'_1)\!)[\underline{\sigma}/\alpha])$ and $(\!(\sigma_2)\!)$ is $\mathsf{fsub}\,((\!(\sigma'_2)\!)[\underline{\sigma}/\alpha])$. Let θ' be $\theta \circ [\underline{\sigma}/\alpha]$. We note that θ' is in $(\!(Q, \alpha = \sigma)\!)$. By induction hypothesis, we have

$$\mathsf{fsub}\left(\theta'((\!(\sigma'_1)\!))\right) = \mathsf{fsub}\left(\theta'((\!(\sigma'_2)\!))\right)$$

Thus, by Property 10.3.1.v, we get the equality

$$\mathsf{fsub}\left(\theta(\mathsf{fsub}\left((\!(\sigma'_1)\!)[\underline{\sigma}/\alpha]\right))\right) = \mathsf{fsub}\left(\theta(\mathsf{fsub}\left((\!(\sigma'_2)\!)[\underline{\sigma}/\alpha]\right))\right)$$

This is exactly $\mathsf{fsub}\,(\theta((\!(\sigma_1)\!))) = \mathsf{fsub}\,(\theta((\!(\sigma_2)\!)))$.

Subcase $\diamond$ is flexible: By definition, we have

$$(\!(\sigma_1)\!) = \mathsf{fsub}\left(\bigcup_{t \in (\!(\sigma)\!)}^{\bar\alpha \# \mathsf{ftv}(\sigma, \sigma'_1)} \forall\,\bar\alpha \cdot (\!(\sigma'_1)\!)[t/\alpha]\right)$$

$$(\!(\sigma_2)\!) = \mathsf{fsub}\left(\bigcup_{t \in (\!(\sigma)\!)}^{\bar\alpha \# \mathsf{ftv}(\sigma, \sigma'_2)} \forall\,\bar\alpha \cdot (\!(\sigma'_2)\!)[t/\alpha]\right)$$

This leads to the following:

$$\mathsf{fsub}\left(\theta(\langle\!\langle\sigma_1\rangle\!\rangle)\right) = \mathsf{fsub}\left(\theta\left(\mathsf{fsub}\left(\bigcup_{t\in\langle\!\langle\sigma\rangle\!\rangle}^{\bar\alpha\#\mathsf{ftv}(\sigma,\sigma_1')} \forall\,\bar\alpha\cdot\langle\!\langle\sigma_1'\rangle\!\rangle[t/\alpha]\right)\right)\right)$$

$$\mathsf{fsub}\left(\theta(\langle\!\langle\sigma_2\rangle\!\rangle)\right) = \mathsf{fsub}\left(\theta\left(\mathsf{fsub}\left(\bigcup_{t\in\langle\!\langle\sigma\rangle\!\rangle}^{\bar\alpha\#\mathsf{ftv}(\sigma,\sigma_2')} \forall\,\bar\alpha\cdot\langle\!\langle\sigma_2'\rangle\!\rangle[t/\alpha]\right)\right)\right)$$

By Property 10.3.1.v, we get

$$\mathsf{fsub}\left(\theta(\langle\!\langle\sigma_1\rangle\!\rangle)\right) = \mathsf{fsub}\left(\theta\left(\bigcup_{t\in\langle\!\langle\sigma\rangle\!\rangle}^{\bar\alpha\#\mathsf{ftv}(\sigma,\sigma_1')} \forall\,\bar\alpha\cdot\langle\!\langle\sigma_1'\rangle\!\rangle[t/\alpha]\right)\right)$$

and

$$\mathsf{fsub}\left(\theta(\langle\!\langle\sigma_2\rangle\!\rangle)\right) = \mathsf{fsub}\left(\theta\left(\bigcup_{t\in\langle\!\langle\sigma\rangle\!\rangle}^{\bar\alpha\#\mathsf{ftv}(\sigma,\sigma_2')} \forall\,\bar\alpha\cdot\langle\!\langle\sigma_2'\rangle\!\rangle[t/\alpha]\right)\right)$$

This can also be written

$$\mathsf{fsub}\left(\theta(\langle\!\langle\sigma_1\rangle\!\rangle)\right) = \bigcup_{t\in\langle\!\langle\sigma\rangle\!\rangle}^{\bar\alpha\#\mathsf{ftv}(\sigma,\sigma_1')} \mathsf{fsub}\left(\theta\left(\forall\,\bar\alpha\cdot\langle\!\langle\sigma_1'\rangle\!\rangle[t/\alpha]\right)\right)$$

$$\mathsf{fsub}\left(\theta(\langle\!\langle\sigma_2\rangle\!\rangle)\right) = \bigcup_{t\in\langle\!\langle\sigma\rangle\!\rangle}^{\bar\alpha\#\mathsf{ftv}(\sigma,\sigma_2')} \mathsf{fsub}\left(\theta\left(\forall\,\bar\alpha\cdot\langle\!\langle\sigma_2'\rangle\!\rangle[t/\alpha]\right)\right)$$

Let J be $\mathsf{ftv}(\sigma,\sigma_1',\sigma_2')\cup\mathsf{dom}(\theta)\cup\mathsf{codom}(\theta)$ **(<u>9</u>)**. By Property ii, $\langle\!\langle\sigma\rangle\!\rangle$ is $\mathsf{ftv}(\sigma,\sigma_2')$-stable, $\langle\!\langle\sigma_1'\rangle\!\rangle$ is $\mathsf{ftv}(\sigma,\sigma_1')$-stable, and $\langle\!\langle\sigma_2'\rangle\!\rangle$ is $\mathsf{ftv}(\sigma,\sigma_2')$-stable. Hence, by Property 10.3.6.ix, we get

$$\mathsf{fsub}\left(\theta(\langle\!\langle\sigma_1\rangle\!\rangle)\right) = \bigcup_{t\in\langle\!\langle\sigma\rangle\!\rangle}^{\bar\alpha\#J} \mathsf{fsub}\left(\theta\left(\forall\,\bar\alpha\cdot\langle\!\langle\sigma_1'\rangle\!\rangle[t/\alpha]\right)\right)\quad(\underline{\mathbf{10}})$$

$$\mathsf{fsub}\left(\theta(\langle\!\langle\sigma_2\rangle\!\rangle)\right) = \bigcup_{t\in\langle\!\langle\sigma\rangle\!\rangle}^{\bar\alpha\#J} \mathsf{fsub}\left(\theta\left(\forall\,\bar\alpha\cdot\langle\!\langle\sigma_2'\rangle\!\rangle[t/\alpha]\right)\right)\quad(\underline{\mathbf{11}})$$

We have to show that these two sets are equal. Let $t \in (\!(\sigma)\!)$ and $\bar\alpha$ disjoint from J $(\underline{\mathbf{12}})$. Let θ' be $\theta \circ [\mathsf{nf}(t)/\alpha]$ $(\underline{\mathbf{13}})$. We note that θ' is in $(\!(Q, \alpha \geq \sigma)\!)$. We have

$$\begin{array}{lllll}
(\underline{\mathbf{14}}) & \mathsf{fsub}\,(\theta'((\!(\sigma'_1)\!))) & = & \mathsf{fsub}\,(\theta'((\!(\sigma'_2)\!))) \\
(\underline{\mathbf{15}}) & \mathsf{fsub}\,(\theta((\!(\sigma'_1)\!)[\mathsf{nf}(t)/\alpha])) & = & \mathsf{fsub}\,(\theta((\!(\sigma'_2)\!)[\mathsf{nf}(t)/\alpha])) \\
(\underline{\mathbf{16}}) & \mathsf{fsub}\,(\theta((\!(\sigma'_1)\!)[t/\alpha])) & = & \mathsf{fsub}\,(\theta((\!(\sigma'_2)\!)[t/\alpha])) \\
(\underline{\mathbf{17}}) & \mathsf{fsub}\,(\forall\,\bar\alpha \cdot \theta((\!(\sigma'_1)\!)[t/\alpha])) & = & \mathsf{fsub}\,(\forall\,\bar\alpha \cdot \theta((\!(\sigma'_2)\!)[t/\alpha])) \\
(\underline{\mathbf{18}}) & \mathsf{fsub}\,(\theta(\forall\,\bar\alpha \cdot (\!(\sigma'_1)\!)[t/\alpha])) & = & \mathsf{fsub}\,(\theta(\forall\,\bar\alpha \cdot (\!(\sigma'_2)\!)[t/\alpha]))
\end{array}$$

Equality (14) holds by induction hypothesis. Equality (15) holds by (13) and (14). Equality (16) holds by (15) by observing that t and $\mathsf{nf}(t)$ are equivalent, thus the set $\mathsf{fsub}\,(\theta((\!(\sigma')\!)[\mathsf{nf}(t)/\alpha]))$ is equal to $\mathsf{fsub}\,(\theta((\!(\sigma')\!)[t/\alpha]))$ for σ' being σ'_1 or σ'_2. Equality (17) holds by Property 10.3.1.iv and (16). Equality (18) holds by commutation of θ and $\forall\,\bar\alpha\cdot$, thanks to (12) and (9). In summary, (18) holds for all t in $(\!(\sigma)\!)$ and $\bar\alpha$ disjoint from J, hence we have $\mathsf{fsub}\,(\theta((\!(\sigma_1)\!))) = \mathsf{fsub}\,(\theta((\!(\sigma_2)\!)))$ from (10) and (11).

∘ Case A-Hyp: We have $(\alpha = \sigma_1) \in Q$ and σ_2 is α. By definition, $(\!(\sigma_2)\!)$ is $\mathsf{fsub}\,(\alpha)$. Necessarily, Q is of the form $(Q_1, \alpha\!=\!\sigma_1, Q_2)$. Hence, $(\!(Q)\!)$ is $(\!(Q_1)\!) \circledcirc (\!(\alpha\!=\!\sigma_1)\!) \circledcirc (\!(Q_2)\!)$. This implies that θ is of the form $\theta_1 \circ \theta' \circ \theta_2$ $(\underline{\mathbf{19}})$, with $\theta_1 \in (\!(Q_1)\!)$, $\theta_2 \in (\!(Q_2)\!)$, and θ' is $[\underline{\sigma_1}/\alpha]$. Hence, we have $\mathsf{fsub}\,(\theta((\!(\sigma_2)\!))) = \mathsf{fsub}\,(\theta(\mathsf{fsub}\,(\alpha)))$, that is, $\mathsf{fsub}\,(\theta((\!(\sigma_2)\!))) = \mathsf{fsub}\,(\theta(\alpha))$ by Property 10.3.1.v. By (19), we get $\mathsf{fsub}\,(\theta((\!(\sigma_2)\!))) = \mathsf{fsub}\,\big(\theta_1(\underline{\sigma_1})\big)$ $(\underline{\mathbf{20}})$. By Property iii, we have $(\!(\sigma_1)\!) = \mathsf{fsub}\,\big(\underline{\sigma_1}\big)$. Hence, $\mathsf{fsub}\,(\theta((\!(\sigma_1)\!))) = \mathsf{fsub}\,\big(\theta(\mathsf{fsub}\,(\underline{\sigma_1}))\big)$. By Property 10.3.1.v, $\mathsf{fsub}\,(\theta((\!(\sigma_1)\!)))$ is $\mathsf{fsub}\,\big(\theta(\underline{\sigma_1})\big)$, that is, $\mathsf{fsub}\,\big(\theta_1(\underline{\sigma_1})\big)$. Then, we get $\mathsf{fsub}\,(\theta((\!(\sigma_1)\!))) = \mathsf{fsub}\,(\theta((\!(\sigma_2)\!)))$ by (20). ∎

Proof of Lemma 10.3.9

By Lemma 2.3.3, we have a restricted thrifty derivation of $(Q)\ \sigma_1 \diamondsuit \sigma_2$ $(\underline{\mathbf{1}})$. Let θ be in $(\!(Q)\!)$. We consider a first case that corresponds to equivalence, then we proceed by case analysis on the last rule in the derivation of (1).

∘ Case $(Q)\ \sigma_1 \equiv \sigma_2$ holds: As in the proof of Lemma 10.3.8, we have $\theta((\!(\sigma_1)\!)) = \theta((\!(\sigma_2)\!))$. This case being solved, we can freely consider that, in the following, $(Q)\ \sigma_1 \equiv \sigma_2$ does not hold.

∘ Case A-Equiv: See the above remark.

∘ Case I-Abstract: By induction hypothesis.

∘ Case R-Trans: By induction hypothesis.

∘ Case R-Context-R: The premise is $(Q, \alpha \diamond \sigma)\ \sigma'_1 \diamondsuit \sigma'_2$ $(\underline{\mathbf{2}})$, σ_1 is $\forall\,(\alpha \diamond \sigma)\ \sigma'_1$, and σ_2 is $\forall\,(\alpha \diamond \sigma)\ \sigma'_2$. We can freely consider $\alpha \notin \mathsf{codom}(\theta)$. If $\mathsf{nf}(\sigma'_1)$ is $\bot$, then $\mathsf{nf}(\sigma_1)$ is $\bot$, thus $(\!(\sigma_1)\!)$ is $(\!(\bot)\!)$ by Property 10.3.6.xii. In this case, $\theta((\!(\sigma_2)\!)) \subseteq \theta((\!(\sigma_1)\!))$ obviously holds. If $\mathsf{nf}(\sigma'_2)$ is $\bot$, then $\mathsf{nf}(\sigma_2)$ is $\bot$, which implies $\mathsf{nf}(\sigma_1) = \bot$ by Property 2.7.7.ii. We solve this case as above. In the following, we consider $\mathsf{nf}(\sigma_1)$ and $\mathsf{nf}(\sigma_2)$ different from $\bot$ $(\underline{\mathbf{3}})$. Since the derivation is thrifty, we have $\mathsf{nf}(\sigma'_1) = \alpha$ if and only if $\mathsf{nf}(\sigma'_2) = \alpha$. We note that if $\mathsf{nf}(\sigma'_1) = \alpha$, then $(\!(\sigma'_1)\!)$ is $\mathsf{fsub}\,(\alpha)$ by Properties 1.5.6.i and 10.3.6.xii. Hence, in this case, $(\!(\sigma_1)\!)$ and $(\!(\sigma_2)\!)$ are equal, which solves this case. In the following, we consider $\mathsf{nf}(\sigma_1)$ and $\mathsf{nf}(\sigma_2)$ different from α $(\underline{\mathbf{4}})$. Additionally, if $\mathsf{nf}(\sigma'_1)$ is a type variable β, then $(Q, \alpha \diamond \sigma)\ \sigma'_1 \equiv \sigma'_2$ holds by Property 2.1.6, thus $(Q)\ \sigma_1 \equiv \sigma_2$ holds. We have already proved such a case. In the following, we consider that $\mathsf{nf}(\sigma_1)$ is not a type variable $(\underline{\mathbf{5}})$. We consider two subcases.

Subcase $\diamond$ is rigid: By definition, $(\!(\sigma_1)\!)$ is the set $\mathsf{fsub}\,((\!(\sigma'_1)\!)[\underline{\sigma}/\alpha])$ $(\underline{\mathbf{6}})$ and $(\!(\sigma_2)\!)$ is the set $\mathsf{fsub}\,((\!(\sigma'_2)\!)[\underline{\sigma}/\alpha])$. Let θ' be $\theta \circ [\underline{\sigma}/\alpha]$. We note that θ' is in $(\!(Q, \alpha = \sigma)\!)$. By induction hypothesis and (2), we have $\theta'((\!(\sigma'_2)\!)) \subseteq \theta'((\!(\sigma'_1)\!))$ $(\underline{\mathbf{7}})$. By Property 10.3.6.vii, (3), (4), and (6), we get $(\!(\sigma_1)\!) =_\alpha (\!(\sigma'_1)\!)[\underline{\sigma}/\alpha]$. By Property 10.3.3.x, we get $\theta((\!(\sigma_1)\!)) =_\alpha \theta'((\!(\sigma'_1)\!))$. Similarly, $\theta((\!(\sigma_2)\!)) =_\alpha \theta'((\!(\sigma'_2)\!))$. From (7), we get $\theta((\!(\sigma_2)\!)) \subseteq_\alpha \theta((\!(\sigma_1)\!))$. This holds for any choice of α not in $\mathsf{codom}(\theta)$, thus, by Property 10.3.3.vi, we get $\theta((\!(\sigma_2)\!)) \subseteq \theta((\!(\sigma_1)\!))$.

Subcase $\diamond$ is flexible: By definition, we have

$$(\!(\sigma_1)\!) = \mathsf{fsub}\left(\bigcup_{t \in (\!(\sigma)\!)}^{\bar\alpha \# \mathsf{ftv}(\sigma, \sigma'_1)} \forall\, \bar\alpha \cdot (\!(\sigma'_1)\!)[t/\alpha]\right)$$

$$(\!(\sigma_2)\!) = \mathsf{fsub}\left(\bigcup_{t \in (\!(\sigma)\!)}^{\bar\alpha \# \mathsf{ftv}(\sigma, \sigma'_2)} \forall\, \bar\alpha \cdot (\!(\sigma'_2)\!)[t/\alpha]\right)$$

Let J be $\mathsf{ftv}(\sigma) \cup \mathsf{ftv}(\sigma'_1) \cup \mathsf{ftv}(\sigma'_2) \cup \mathsf{dom}(\theta) \cup \mathsf{codom}(\theta)$. By Properties 10.3.7.ii and 10.3.6.ix, we get

$$(\!(\sigma_1)\!) = \mathsf{fsub}\left(\bigcup_{t \in (\!(\sigma)\!)}^{\bar\alpha \# J} \forall\, \bar\alpha \cdot (\!(\sigma'_1)\!)[t/\alpha]\right) \quad (\underline{\mathbf{8}})$$

$$(\!(\sigma_2)\!) = \mathsf{fsub}\left(\bigcup_{t \in (\!(\sigma)\!)}^{\bar\alpha \# J} \forall\, \bar\alpha \cdot (\!(\sigma'_2)\!)[t/\alpha]\right) \quad (\underline{\mathbf{9}})$$

Let t_2 be in $\theta((\!(\sigma_2)\!))$. We note that $\mathsf{ftv}(t_2)\ \#\ \mathsf{dom}(\theta)$ $(\underline{\mathbf{10}})$ holds. By (9), there exist t in $(\!(\sigma)\!)$ and $\bar\alpha$ disjoint from J such that t_2 is in $\theta(\mathsf{fsub}\,(\forall\, \bar\alpha \cdot (\!(\sigma'_2)\!)[t/\alpha]))$. By Property 10.3.1.ii, we have t_2 in the set $\mathsf{fsub}\,(\theta(\forall\, \bar\alpha \cdot (\!(\sigma'_2)\!)[t/\alpha]))$. Since $\bar\alpha$ is disjoint from J, this set is equal to $\mathsf{fsub}\,(\forall\, \bar\alpha \cdot \theta \circ [t/\alpha]((\!(\sigma'_2)\!)))$. This set is closed by equivalence in System F (it is of the form $\mathsf{fsub}\,(S)$), hence it is equal to $\mathsf{fsub}\,(\forall\, \bar\alpha \cdot \theta \circ [\mathsf{nf}(t)/\alpha]((\!(\sigma'_2)\!)))$. Let θ' be $\theta \circ [\mathsf{nf}(t)/\alpha]$. We note that θ' is in $(\!(Q, \alpha \geq \sigma)\!)$. Hence, by induction hypothesis, we have $\theta'((\!(s'_2)\!)) \subseteq \theta'((\!(\sigma'_1)\!))$. As a consequence, t_2 is in $\mathsf{fsub}\,(\forall\, \bar\alpha \cdot \theta \circ [\mathsf{nf}(t)/\alpha]((\!(\sigma'_1)\!)))$, that is, t in $\mathsf{fsub}\left(\theta(\forall\,\bar\alpha\cdot(\!(\sigma'_1)\!)[\mathsf{nf}(t)/\alpha])\right)$. For any t_1 in $\forall\,\bar\alpha\cdot(\!(\sigma'_1)\!)[\mathsf{nf}(t)/\alpha]$, we have $t_1/\epsilon = \sigma'_1/e$ by Property 10.3.6.vi, (3), and (4). In particular, (5) implies that the normal form of t_1 is not in the set $\mathsf{dom}(\theta)$. By Property 10.3.6.v and (10), the type t_2 is in the set $\theta(\mathsf{fsub}\left(\forall\,\bar\alpha\cdot(\!(\sigma'_1)\!)[\mathsf{nf}(t)/\alpha]\right))$. Since t and $\mathsf{nf}(t)$ are equivalent, we also have t_2 in $\theta(\mathsf{fsub}\left(\forall\,\bar\alpha\cdot(\!(\sigma'_1)\!)[t/\alpha]\right))$. Hence, t_2 is in

$$\theta(\mathsf{fsub}\left(\bigcup_{t\in(\!(\sigma)\!)}^{\bar\alpha\# J}\forall\,\bar\alpha\cdot(\!(\sigma'_1)\!)[t/\alpha]\right))$$

By (8), that is $t_2 \in \theta((\!(\sigma_1)\!))$ In summary, we have shown $\theta((\!(\sigma_2)\!)) \subseteq \theta((\!(\sigma_1)\!))$.

∘ Case R-Context-Rigid: The premise is $(Q)\ \sigma'_1 \sqsubseteq\!\!\!= \sigma'_2$ **(<u>11</u>)**, σ_1 is $\forall\,(\alpha = \sigma'_1)\ \sigma$, and σ_2 is $\forall\,(\alpha = \sigma'_2)\ \sigma$. We can freely assume $\alpha \notin \mathsf{codom}(\theta)$. By definition, $(\!(\sigma_1)\!)$ is $\mathsf{fsub}\left((\!(\sigma)\!)[\underline{\sigma'_1}/\alpha]\right)$ and $(\!(\sigma_2)\!)$ is $\mathsf{fsub}\left((\!(\sigma)\!)[\underline{\sigma'_2}/\alpha]\right)$. By restrictions of Lemma 2.3.1, we have $\mathsf{nf}(\sigma) \neq \alpha$ and $\mathsf{nf}(\sigma) \neq \bot$, which imply $(\!(\sigma_1)\!) =_\alpha (\!(\sigma)\!)[\underline{\sigma'_1}/\alpha]$ and $(\!(\sigma_2)\!) =_\alpha (\!(\sigma)\!)[\underline{\sigma'_2}/\alpha]$ by Property 10.3.6.vii (page 196). By Property 10.3.3.x (page 194), we get $\theta((\!(\sigma_1)\!)) =_\alpha \theta((\!(\sigma)\!)[\underline{\sigma'_1}/\alpha])$ and $\theta((\!(\sigma_2)\!)) =_\alpha \theta((\!(\sigma)\!)[\underline{\sigma'_2}/\alpha])$. This is equivalent to $\theta((\!(\sigma_1)\!)) =_\alpha \theta((\!(\sigma)\!)[\theta(\underline{\sigma'_1})/\alpha])$ **(<u>12</u>)** and $\theta((\!(\sigma_2)\!)) =_\alpha \theta((\!(\sigma)\!)[\theta(\underline{\sigma'_2})/\alpha])$ **(<u>13</u>)**. By Lemma 10.3.8 (page 197) and (11), we have $\mathsf{fsub}\,(\theta((\!(\sigma'_1)\!))) = \mathsf{fsub}\,(\theta((\!(\sigma'_2)\!)))$. By Property 10.3.7.iii (page 196), it gives

$$\mathsf{fsub}\left(\theta(\mathsf{fsub}\left(\underline{\sigma'_1}\right))\right) = \mathsf{fsub}\left(\theta(\mathsf{fsub}\left(\underline{\sigma'_2}\right))\right)$$

By Property 10.3.1.v, we get $\mathsf{fsub}\left(\theta(\underline{\sigma'_1})\right) = \mathsf{fsub}\left(\theta(\underline{\sigma'_2})\right)$ **(<u>14</u>)**. In particular, we have $\theta(\underline{\sigma'_1}) \in \mathsf{fsub}\left(\theta(\underline{\sigma'_1})\right)$. Hence, by (14), we have $\theta(\underline{\sigma'_1}) \in \mathsf{fsub}\left(\theta(\underline{\sigma'_2})\right)$, which means that $\theta(\underline{\sigma'_2}) \sqsubseteq_F \theta(\underline{\sigma'_1})$ **(<u>15</u>)** holds. Similarly, we show that $\theta(\underline{\sigma'_1}) \sqsubseteq_F \theta(\underline{\sigma'_2})$ **(<u>16</u>)** holds. As a consequence of (15) and (16), $\theta(\underline{\sigma'_1})$ and $\theta(\underline{\sigma'_2})$ are equivalent in System F. Observing that $\underline{\sigma'_1}$ and $\underline{\sigma'_2}$ are in normal form by definition, we get $\theta(\underline{\sigma'_1}) = \theta(\underline{\sigma'_2})$. By (12) and (13), it leads to $\theta((\!(\sigma_1)\!)) =_\alpha \theta((\!(\sigma_1)\!))$. This holds for any choice of α such that $\alpha \notin \mathsf{codom}(\theta)$. Hence, by Property 10.3.3.vii, we get $\theta((\!(\sigma_1)\!)) = \theta((\!(\sigma_2)\!))$.

∘ Case R-Context-Flexible: The premise is $(Q)\ \sigma'_1 \sqsubseteq \sigma'_2$ **(<u>17</u>)**, σ_1 is $\forall\,(\alpha \geq \sigma'_1)$ σ, and σ_2 is $\forall\,(\alpha \geq \sigma'_2)\ \sigma$. By induction hypothesis and (17), we get $\theta((\!(\sigma'_2)\!)) \subseteq \theta((\!(\sigma'_1)\!))$ **(<u>18</u>)**. By definition, we have

$$(\!(\sigma_1)\!) = \mathsf{fsub}\left(\bigcup_{t'_1\in(\!(\sigma'_1)\!)}^{\bar\alpha\#\mathsf{ftv}(\sigma'_1,\sigma)}\forall\,\bar\alpha\cdot(\!(\sigma)\!)[t'_1/\alpha]\right)$$

$$(\!(\sigma_2)\!) = \mathsf{fsub}\left(\bigcup_{t'_2\in(\!(\sigma'_2)\!)}^{\bar\alpha\#\mathsf{ftv}(\sigma'_2,\sigma)}\forall\,\bar\alpha\cdot(\!(\sigma)\!)[t'_2/\alpha]\right)$$

Let J be $\mathsf{ftv}(\sigma, \sigma'_1, \sigma'_2) \cup \mathsf{dom}(\theta) \cup \mathsf{codom}(\theta)$. By Properties 10.3.7.ii and 10.3.6.ix, we get

$$(\!(\sigma_1)\!) = \mathsf{fsub}\left(\bigcup_{t'_1 \in (\!(\sigma'_1)\!)}^{\bar\alpha \# J} \forall\,\bar\alpha \cdot (\!(\sigma)\!)[t'_1/\alpha]\right)$$

$$(\!(\sigma_2)\!) = \mathsf{fsub}\left(\bigcup_{t'_2 \in (\!(\sigma'_2)\!)}^{\bar\alpha \# J} \forall\,\bar\alpha \cdot (\!(\sigma)\!)[t'_2/\alpha]\right)$$

Applying θ, we get

$$\theta((\!(\sigma_1)\!)) = \theta\left(\mathsf{fsub}\left(\bigcup_{t'_1 \in (\!(\sigma'_1)\!)}^{\bar\alpha \# J} \forall\,\bar\alpha \cdot (\!(\sigma)\!)[t'_1/\alpha]\right)\right)$$

$$\theta((\!(\sigma_2)\!)) = \theta\left(\mathsf{fsub}\left(\bigcup_{t'_2 \in (\!(\sigma'_2)\!)}^{\bar\alpha \# J} \forall\,\bar\alpha \cdot (\!(\sigma)\!)[t'_2/\alpha]\right)\right)$$

By restrictions of Lemma 2.3.1, we have $\alpha \in \mathsf{ftv}(\sigma)$, and $\mathsf{nf}(\sigma) \neq \alpha$. As a consequence, σ/ϵ is a type constructor g. Hence, for all t in $(\!(\sigma)\!)$, for all $\bar\alpha$ and all $t'_1 \in (\!(\sigma'_1)\!)$, $\forall\,\bar\alpha \cdot t[t'_1/\alpha]/\epsilon = g$. In particular, $t \notin \mathsf{dom}(\theta)$. By Property 10.3.6.v, we get

$$\theta((\!(\sigma_1)\!)) =_{\mathsf{dom}(\theta)} \mathsf{fsub}\left(\bigcup_{t'_1 \in (\!(\sigma'_1)\!)}^{\bar\alpha \# J} \theta(\forall\,\bar\alpha \cdot (\!(\sigma)\!)[t'_1/\alpha])\right)$$

$$\theta((\!(\sigma_2)\!)) =_{\mathsf{dom}(\theta)} \mathsf{fsub}\left(\bigcup_{t'_2 \in (\!(\sigma'_2)\!)}^{\bar\alpha \# J} \theta(\forall\,\bar\alpha \cdot (\!(\sigma)\!)[t'_2/\alpha])\right)$$

Hence, we have

$$\theta((\!(\sigma_1)\!)) =_{\mathsf{dom}(\theta)} \mathsf{fsub}\left(\bigcup_{t'_1 \in (\!(\sigma'_1)\!)}^{\bar\alpha \# J} \forall\,\bar\alpha \cdot \theta((\!(\sigma)\!))[\theta(t'_1)/\alpha]\right)$$

$$\theta((\!(\sigma_2)\!)) =_{\mathsf{dom}(\theta)} \mathsf{fsub}\left(\bigcup_{t'_2 \in (\!(\sigma'_2)\!)}^{\bar\alpha \# J} \forall\,\bar\alpha \cdot \theta((\!(\sigma)\!))[\theta(t'_2)/\alpha]\right)$$

We note that the sets $\{\theta(t'_1) \mid t'_1 \in \llparenthesis \sigma'_1 \rrparenthesis\}$ and $\{t'_1 \mid t'_1 \in \theta(\llparenthesis \sigma'_1 \rrparenthesis)\}$ are equal. Hence, we have

$$\theta(\llparenthesis \sigma_1 \rrparenthesis) =_{\mathsf{dom}(\theta)} \mathsf{fsub}\left(\bigcup_{t'_1 \in \theta(\llparenthesis \sigma'_1 \rrparenthesis)}^{\bar{\alpha} \# J} \forall\, \bar{\alpha} \cdot \theta(\llparenthesis \sigma \rrparenthesis)[t'_1/\alpha]\right)$$

$$\theta(\llparenthesis \sigma_2 \rrparenthesis) =_{\mathsf{dom}(\theta)} \mathsf{fsub}\left(\bigcup_{t'_2 \in \theta(\llparenthesis \sigma'_2 \rrparenthesis)}^{\bar{\alpha} \# J} \forall\, \bar{\alpha} \cdot \theta(\llparenthesis \sigma \rrparenthesis)[t'_2/\alpha]\right)$$

By (18), we get $\theta(\llparenthesis \sigma_2 \rrparenthesis) \subseteq_{\mathsf{dom}(\theta)} \theta(\llparenthesis \sigma_1 \rrparenthesis)$. We note that θ is idempotent, thus $\theta(\llparenthesis \sigma_i \rrparenthesis) \cap \Sigma^{\neg}_{\mathsf{dom}(\theta)} = \theta(\llparenthesis \sigma_i \rrparenthesis)$ for σ_i being σ_1 or σ_2. Hence, $\theta(\llparenthesis \sigma_2 \rrparenthesis) \subseteq \theta(\llparenthesis \sigma_1 \rrparenthesis)$ holds.

∘ CASE A-HYP: We have $(\alpha = \sigma_1) \in Q$ and σ_2 is α. By definition, $\llparenthesis \sigma_2 \rrparenthesis$ is $\mathsf{fsub}\,(\alpha)$. Necessarily, Q is of the form $(Q_1, \alpha = \sigma_1, Q_2)$. Hence, $\llparenthesis Q \rrparenthesis$ is $\llparenthesis Q_1 \rrparenthesis \circledcirc \llparenthesis \alpha = \sigma_1 \rrparenthesis \circledcirc \llparenthesis Q_2 \rrparenthesis$. This implies that θ is of the form $\theta_1 \circ [\underline{\sigma_1}/\alpha] \circ \theta_2$, with $\theta_1 \in \llparenthesis Q_1 \rrparenthesis$ and $\theta_2 \in \llparenthesis Q_2 \rrparenthesis$. Hence, all types in $\theta(\llparenthesis \sigma_2 \rrparenthesis)$ are equivalent to $\theta_1(\underline{\sigma_1})$, that is, $\theta(\underline{\sigma_1})$. This implies $\theta(\llparenthesis \sigma_2 \rrparenthesis) \subseteq \mathsf{fsub}\,\big(\theta(\underline{\sigma_1})\big)$. By Property 10.3.1.v, we get $\theta(\llparenthesis \sigma_2 \rrparenthesis) \subseteq \mathsf{fsub}\,\big(\theta(\mathsf{fsub}\,(\underline{\sigma_1}))\big)$. By Property 10.3.7.iii, we have $\theta(\llparenthesis \sigma_2 \rrparenthesis) \subseteq \mathsf{fsub}\,(\theta(\llparenthesis \sigma_1 \rrparenthesis))$. The restrictions of Lemma 2.3.1 ensure that σ_1 is not in $\mathcal{T}$. Hence, $\mathsf{nf}(\sigma_1)$ is not in $\mathsf{dom}(\theta)$. As a consequence, we have $\theta(\llparenthesis \sigma_2 \rrparenthesis) \subseteq_{\mathsf{dom}(\theta)} \theta(\mathsf{fsub}\,(\llparenthesis \sigma_1 \rrparenthesis))$. By Property 10.3.6.i, we get $\theta(\llparenthesis \sigma_2 \rrparenthesis) \subseteq_{\mathsf{dom}(\theta)} \theta(\llparenthesis \sigma_1 \rrparenthesis)$. Observing that θ is idempotent, we get $\theta(\llparenthesis \sigma_2 \rrparenthesis) \cap \Sigma^{\neg}_{\mathsf{dom}(\theta)} = \theta(\llparenthesis \sigma_2 \rrparenthesis)$, thus $\theta(\llparenthesis \sigma_2 \rrparenthesis) \subseteq_{\mathsf{dom}(\theta)} \theta(\llparenthesis \sigma_1 \rrparenthesis)$.

∘ CASE I-HYP: We have $(\alpha \geq \sigma_1) \in Q$ and σ_2 is α. By definition, $\llparenthesis \sigma_2 \rrparenthesis$ is $\mathsf{fsub}\,(\alpha)$. Necessarily, Q is of the form $(Q_1, \alpha \geq \sigma_1, Q_2)$. Hence, $\llparenthesis Q \rrparenthesis$ is $\llparenthesis Q_1 \rrparenthesis \circledcirc \llparenthesis \alpha \geq \sigma_1 \rrparenthesis \circledcirc \llparenthesis Q_2 \rrparenthesis$. This implies that θ is of the form $\theta_1 \circ [t/\alpha] \circ \theta_2$, with $\theta_1 \in \llparenthesis Q_1 \rrparenthesis$, $\theta_2 \in \llparenthesis Q_2 \rrparenthesis$, and $t \in \llparenthesis \sigma_1 \rrparenthesis$. Hence, all types in $\theta(\llparenthesis \sigma_2 \rrparenthesis)$ are equivalent to $\theta_1(t)$, that is, $\theta(t)$. We conclude as in the case A-HYP.

∘ CASE I-BOT: We have $\sigma_1 = \bot$, and observing that $\llparenthesis \bot \rrparenthesis$ is $\mathsf{fsub}\,(\forall\, \alpha \cdot \alpha)$, we have $\theta(\llparenthesis \sigma_2 \rrparenthesis) \subseteq \theta\llparenthesis \bot \rrparenthesis$.

∘ CASE I-RIGID: We have $\sigma_1 = \forall\,(\alpha \geq \sigma)\ \sigma'$, and $\sigma_2 = \forall\,(\alpha = \sigma)\ \sigma'$. By definition, $\llparenthesis \sigma_2 \rrparenthesis$ is $\mathsf{fsub}\,(\llparenthesis \sigma' \rrparenthesis[\underline{\sigma}/\alpha])$, and $\llparenthesis \sigma_1 \rrparenthesis$ is $\mathsf{fsub}\left(\bigcup_{t \in \llparenthesis \sigma \rrparenthesis}^{\bar{\alpha} \# \mathsf{ftv}(\sigma, \sigma')} \forall\, \bar{\alpha} \cdot \llparenthesis \sigma' \rrparenthesis[t/\alpha]\right)$ (**<u>19</u>**) By Property 10.3.7.iii, we have $\underline{\sigma} \in \llparenthesis \sigma \rrparenthesis$. Let t be $\underline{\sigma}$. Taking $\bar{\alpha} = \emptyset$ in (19), we immediately have $\mathsf{fsub}\,(\llparenthesis \sigma' \rrparenthesis[t/\alpha]) \subseteq \llparenthesis \sigma_1 \rrparenthesis$, that is, $\llparenthesis \sigma_2 \rrparenthesis \subseteq \llparenthesis \sigma_1 \rrparenthesis$. This implies $\theta(\llparenthesis \sigma_2 \rrparenthesis) \subseteq \theta(\llparenthesis \sigma_1 \rrparenthesis)$.
■

Indexes

Defined rules

Many inference rules are introduced in the document. They are listed in the following table. Names of derived rules are followed by a star * and can be found page 321.

Rule name	Relation	Page [fig]
A-Alias'	$⋿^{\bar{\alpha}}$	p.82 [2.1]
A-Context*	$⋿$	p.77
A-Equiv	$⋿$	p.56 [1.2]
A-Hyp	$⋿$	p.56 [1.2]
A-Up*	$⋿$	p.56
App$^{\triangledown}$	$\vdash^{\triangledown}$	p.149 [5.3]
C-Abstract-F	$\dot{\sqsubseteq}$	p.85 [2.5]
C-Strict	$\dot{\sqsubseteq}$	p.85 [2.5]
Eq-Context*	$\equiv$	p.77
Eq-Mono*	$\equiv$	p.47
Eq-Refl	$\equiv$	p.46 [1.1]
Eq-Var	$\equiv$	p.46 [1.1]
Fun*	$\vdash$	p.176
Gen	$\vdash$	p.145 [5.2]
I-Abstract	$\sqsubseteq$	p.59 [1.3]
I-Bot'	$\sqsubseteq^{\bar{\alpha}}$	p.83 [2.2]
I-Context-L'	$\sqsubseteq^{\bar{\alpha}}$	p.83 [2.2]
continued...		

Rule name	Relation	Page [fig]
A-Context-L'	$⋿^{\bar{\alpha}}$	p.82 [2.1]
A-Equiv'	$⋿^{\bar{\alpha}}$	p.82 [2.1]
A-Hyp'	$⋿^{\bar{\alpha}}$	p.82 [2.1]
A-Up'	$⋿^{\bar{\alpha}}$	p.82 [2.1]
Ac-Hyp	$⋿_C$	p.78
App	$\vdash$	p.145 [5.2]
C-Abstract-R	$\dot{\sqsubseteq}$	p.85 [2.5]
Eq-Comm	$\equiv$	p.46 [1.1]
Eq-Free	$\equiv$	p.46 [1.1]
Eq-Mono	$\equiv$	p.46 [1.1]
Eq-Var*	$\equiv$	p.48
Fun$^{\triangledown}$	$\vdash^{\triangledown}$	p.149 [5.3]
Fun	$\vdash$	p.145 [5.2]
I-Abstract'	$\sqsubseteq^{\bar{\alpha}}$	p.83 [2.2]
I-Alias'	$\sqsubseteq^{\bar{\alpha}}$	p.83 [2.2]
I-Bot	$\sqsubseteq$	p.59 [1.3]
I-Context*	$\sqsubseteq$	p.77
continued...		

Rule name	Relation	Page [fig]
I-Drop*	$\sqsubseteq$	p.58
I-Hyp'	$\sqsubseteq^{\bar{\alpha}}$	p.83 [2.2]
I-Rigid'	$\sqsubseteq^{\bar{\alpha}}$	p.83 [2.2]
I-Up'	$\sqsubseteq^{\bar{\alpha}}$	p.83 [2.2]
Ic-Abstract	$\sqsubseteq_C$	p.79
Ic-Hyp	$\sqsubseteq_C$	p.79
Inst	$\vdash$	p.145 [5.2]
Let	$\vdash$	p.145 [5.2]
Oracle	$\vdash$	p.145 [5.2]
PA-Equiv	$\mathrel{\sqsubset\!\!\!\!-}_\ell^I$	p.105 [3.2]
PE-Comm	$\equiv_\ell^I$	p.105 [3.1]
PE-Free	$\equiv_\ell^I$	p.105 [3.1]
PE-Mono	$\equiv_\ell^I$	p.105 [3.1]
PE-Swap	$\equiv_\ell^I$	p.105 [3.1]
PI-Abstract	$\sqsubseteq_\ell^I$	p.106 [3.3]
PI-Rigid	$\sqsubseteq_\ell^I$	p.106 [3.3]
Pr-Subst*	$\Diamond$	p.102
R-Context-Rigid	$\Diamond$	p.46
R-Context-R	$\Diamond$	p.45
S-Alias	$\dot{\sqsubset}$	p.84 [2.3]
S-Nil	$\dot{\sqsubset}$	p.84 [2.3]
S-Up	$\dot{\sqsubset}$	p.84 [2.3]
StSh-Hyp	$\dot{\mathrel{\sqsubset\!\!\!\!-}}^{\bar{\alpha}}$	p.85 [2.4]
Strengthen'	$\vdash'$	p.155 –6.2.3
Var$^\triangledown$	$\vdash^\triangledown$	p.149 –5.3
Weaken*	$\vdash$	p.152 –6.1.2

Rule name	Relation	Page [fig]
I-Equiv*	$\sqsubseteq$	p.58
I-Hyp	$\sqsubseteq$	p.59 [1.3]
I-Rigid	$\sqsubseteq$	p.59 [1.3]
I-Up*	$\sqsubseteq$	p.58
Ic-Bot	$\sqsubseteq_C$	p.79
Ic-Rigid	$\sqsubseteq_C$	p.79
Let$^\triangledown$	$\vdash^\triangledown$	p.149 [5.3]
Oracle$^\triangledown$	$\vdash^\triangledown$	p.149 [5.3]
PA-Context-L	$\mathrel{\sqsubset\!\!\!\!-}_\ell^I$	p.105 [3.2]
PA-Trans	$\mathrel{\sqsubset\!\!\!\!-}_\ell^I$	p.105 [3.2]
PE-Context-L	$\equiv_\ell^I$	p.105 [3.1]
PE-Mono*	$\equiv_\ell^I$	p.106
PE-Refl	$\equiv_\ell^I$	p.105 [3.1]
PE-Trans	$\equiv_\ell^I$	p.105 [3.1]
PI-Context-L	$\sqsubseteq_\ell^I$	p.106 [3.3]
PI-Trans	$\sqsubseteq_\ell^I$	p.106 [3.3]
R-Context-Flexible	$\Diamond$	p.46
R-Context-L	$\Diamond$	p.46
R-Trans	$\Diamond$	p.45
S-Hyp	$\dot{\sqsubset}$	p.84 [2.3]
S-Rigid	$\dot{\sqsubset}$	p.84 [2.3]
StSh-Alias	$\dot{\mathrel{\sqsubset\!\!\!\!-}}^{\bar{\alpha}}$	p.85 [2.4]
StSh-Up	$\dot{\mathrel{\sqsubset\!\!\!\!-}}^{\bar{\alpha}}$	p.85 [2.4]
Strengthen	$\vdash$	p.152 –6.1.2
Var	$\vdash$	p.145 [5.2]
Weaken	$\vdash$	p.152 6.1.2

Derivable rules

Some of the rules in the previous table are actually derivable rules. We explicitly give such rules here.

Name	*Rule*	Page
Eq-Mono*	$\dfrac{(Q)\ \sigma' \equiv \tau}{(Q)\ \forall\,(\alpha \diamond \sigma')\ \sigma \equiv \sigma[\tau/\alpha]}$	P.47
Eq-Var*	$\dfrac{(\alpha \diamond \sigma) \in Q}{(Q_0)\ \forall\,(Q)\ \alpha \equiv \forall\,(Q)\ \sigma}$	P.48
A-Up*	$\dfrac{\alpha' \notin \mathsf{ftv}(\sigma_0)}{\forall\,(\alpha = \forall\,(\alpha' = \sigma')\ \sigma)\ \sigma_0 \mathrel{-\mkern-12mu\sqsubseteq} \forall\,(\alpha' = \sigma')\ \forall\,(\alpha = \sigma)\ \sigma_0}$	P.56
I-Drop*	$\dfrac{}{(QQ'Q'')\ \forall\,(Q')\ \sigma \sqsubseteq \sigma}$	P.58
I-Equiv*	$\dfrac{(Q)\ \sigma_1 \equiv \sigma_2}{(Q)\ \sigma_1 \sqsubseteq \sigma_2}$	P.58
I-Up*	$\dfrac{\alpha_2 \notin \mathsf{ftv}(\sigma)}{(Q)\ \forall\,(\alpha_1 \geq \forall\,(\alpha_2 \diamond \sigma_2)\ \sigma_1)\ \sigma \sqsubseteq \forall\,(\alpha_2 \diamond \sigma_2)\ \forall\,(\alpha_1 \geq \sigma_1)\ \sigma}$	P.58
Eq-Context*	$\dfrac{(Q\overline{C})\ \sigma_1 \equiv \sigma_2}{(Q)\ C(\sigma_1) \equiv C(\sigma_2)}$	P.77
A-Context*	$\dfrac{(Q\overline{C_r})\ \sigma_1 \mathrel{-\mkern-12mu\sqsubseteq} \sigma_2}{(Q)\ C_r(\sigma_1) \mathrel{-\mkern-12mu\sqsubseteq} C_r(\sigma_2)}$	P.77

continued...

Name	*Rule*	Page
I-Context*	$\dfrac{(Q\overline{C_f})\ \sigma_1 \sqsubseteq \sigma_2}{(Q)\ C_f(\sigma_1) \sqsubseteq C_f(\sigma_2)}$	P.77
Pr-Subst*	$\dfrac{(Q\underline{\theta})\ \sigma_1 \diamondsuit \sigma_2}{(Q)\ \theta(\sigma_1) \diamondsuit \theta(\sigma_2)}$	P.102
PE-Mono*	$\dfrac{(Q)\ \sigma \equiv \tau}{(Q, \alpha = \sigma, Q') \equiv_\ell (Q, \alpha = \sigma, Q'[\tau/\alpha])}$	P.106

Notations

The following table lists the symbols and notations used in this document. The left column contains a symbol or a word. Then comes the page where it is defined. In most cases, the third column provides the corresponding definition, or it simply reads the symbol.

In a few cases, the prefix Q is supposed equal to $(\alpha_1 \diamond_1 \sigma_1, \ldots \alpha_n \diamond_n \sigma_n)$ **($\underline{1}$)**. We explicitly refer to this definition, then.

Symbol	Page	Meaning
α, τ, σ	38	A type variable, a monotype, a polytype
$\bar{\alpha}$, $\bar{\tau}$	39	A tuple of type variables, a tuple of monotypes
$Q_1 \# Q_2$	39	$\mathsf{dom}(Q_1) \# \mathsf{dom}(Q_2)$
$\bar{\alpha} \# \bar{\beta}$	39	$\bar{\alpha} \cap \bar{\beta} = \emptyset$
ϕ disjoint from $\bar{\alpha}$	42	$\bar{\alpha} \# \mathsf{dom}(\phi) \cup \mathsf{codom}(\phi)$
$\#(\sigma)$	97	The cardinal of $\mathsf{dom}(\sigma)$
$\star$	91	Binary operator on $\{X, Y, Z\}$
$\forall\,(Q)\ \sigma$	39	$\forall\,(\alpha_1 \diamond_1 \sigma_1) \ldots \forall\,(\alpha_n \diamond_n \sigma_n)\ \sigma$ when Q is (1)

continued. . .

Symbol	Page	Meaning
$\forall\,\alpha\cdot S$	192	$\{\forall\,\alpha\cdot t \mid t\in S\}$
$\sqsubseteq$	57	The instance relation, figure 1.3
$\dot{\sqsubseteq}$	85	See figure 2.5
$\dot{\sqsubset}$	84	See figure 2.3
$\sqsubseteq^I$	103	Prefix instance relation
$\sqsubseteq^I_\ell$	106	See figure 3.3
$\sqsubseteq_C$	79	
$\sqsubseteq_F$	192	The instance relation of System F.
$\sqsubseteq^{\bar\alpha}$	83	See figure 2.2
$\mathrel{-\!\!\!\!\sqsubseteq}$	55	The abstraction relation, figure 1.2
$\dot{\mathrel{-\!\!\!\!\sqsubseteq}}^{\bar\alpha}$	85	See figure 2.4
$(Q)\ \sigma_1 \mathrel{-\!\!\!\!\sqsubseteq}^{?} \sigma_2$	121	The abstraction-check algorithm
$\mathrel{-\!\!\!\!\sqsubseteq}^I$	103	Prefix abstraction relation
$\mathrel{-\!\!\!\!\sqsubseteq}^I_\ell$	105	See figure 3.2
$\mathrel{-\!\!\!\!\sqsubseteq}_C$	78	
$\mathrel{-\!\!\!\!\sqsubseteq}^{\bar\alpha}$	82	See figure 2.1
ϵ	40	The empty sequence
$\approx$	48	The rearrangement relation, Definition 1.5.2
$S_1 =_{\bar\alpha} S_2$	194	$S_1\cap\Sigma_{\bar\alpha}^{\neg} = S_2\cap\Sigma_{\bar\alpha}^{\neg}$
$\equiv$	47	The equivalence relation, figure 1.1
$(\equiv\dot{\mathrel{-\!\!\!\!\sqsubseteq}}^{\emptyset})$, $(\equiv\dot{\sqsubseteq})$	84	See Definition 2.6.1
$(\equiv\mid\sqsubseteq_C)$	79	
$(\equiv\mid\mathrel{-\!\!\!\!\sqsubseteq}_C)$	78	
$\equiv^I$	103	Prefix equivalence relation
$\equiv^I_\ell$	105	See figure 3.1
$\diamond$, $\diamond'$, $\diamond_1$	39	Meta-variables standing for $=$ or $\geq$, as in *e.g.* $(\alpha_1 \diamond_1 \sigma_1, \alpha_2 \diamond_2 \sigma_2)$
$\mathcal{T}$	51	The set $\{\sigma \mid \exists\tau.\mathsf{nf}(\sigma)=\tau\}$

continued...

Symbol	Page	Meaning
$\bot$	38	“bottom”, equivalent to $\forall\,(\alpha)\;\alpha$
ϕ	42	An idempotent renaming
$\phi^{\neg}$	42	The inverse renaming of ϕ
$\phi(Q)$	42	$(\phi(\alpha_1)\diamond_1\phi(\sigma_1)\ldots\phi(\alpha_n)\diamond_n\phi(\sigma_n))$ when Q is (1)
$\Diamond$	45	Meta-variable standing for $\equiv$, $\mathrel{\sqsubset\!\!\!\!-}$, or $\sqsubseteq$, as in *e.g.* $(Q)\;\sigma_1 \Diamond \sigma_2$
$Q \Diamond Q'$	103	$Q \Diamond^{\mathsf{dom}(Q)} Q'$
$\sigma \cdot u/$	40	The function $u' \mapsto \sigma/uu'$
$\lceil\sigma\rceil$	128	See Definition 4.5.2
$\sigma'[\sigma/\alpha]$	42	The substitution $[\sigma/\alpha]$ applied to the type σ'
$\sigma/$	40	The function mapping occurrences in σ to type symbols
σ/u	40	The type symbol at occurrence u in σ
$\underline{\sigma}$	192	The projection of an F-type σ to a type of System F.
$[\sigma/\alpha]$	42	The capture-avoiding substitution of α by σ
Σ_I	103	The set $\{\sigma \mid \mathsf{utv}(\sigma) \subseteq I\}$
$\Sigma_{\bar\alpha}^{\neg}$	194	The set of System F types $\{t \mid \mathsf{ftv}(\sigma)\ \#\ \bar\alpha\}$
$P \leqslant P'$ (polynomials)	86	
$\leqslant_{/}$	65	A partial order on skeletons
$S_1 \subseteq_{\bar\alpha} S_2$	194	$S_1 \cap \Sigma_{\bar\alpha}^{\neg} \subseteq S_2 \cap \Sigma_{\bar\alpha}^{\neg}$
$\vdash$	145	Typing judgments, see figure 5.2
$\vdash^{\triangledown}$	149	Syntax-directed typing judgments, see figure 5.3
Θ	40	A substitution on skeletons
θ	42	An idempotent substitution
$\theta(S)$	192	$\{\theta(t) \mid t \in S\}$
$\theta(Q)$	42	

continued…

Symbol	Page	Meaning
$\underline{\theta}$	102	The monotype prefix corresponding to θ
Θ_Q	40	Assuming Q is (1), this is the substitution $[\mathsf{proj}(\sigma_1)/\alpha_1] \circ .. \circ [\mathsf{proj}(\sigma_n)/\alpha_n]$
$[\,]$	71	A hole (in contexts)
∇_Q	39	$\nabla_{\mathsf{dom}(Q)}$
$\nabla_{\alpha_1,\ldots\alpha_n}$	39	$\alpha_1 \rightarrow \ldots \rightarrow \alpha_n \rightarrow \mathtt{unit}$
C	71	Generic contexts
$\overline{C}$	71	The prefix corresponding to C
$\widehat{C}$	71	The extracted substitution $\widehat{\overline{C}}$
C^n	76	Contexts with n holes
C_f	74	Flexible contexts
C_n	73	Narrow contexts
C_r	74	Rigid contexts
$\mathsf{cf}(\sigma)$	55	The constructed form of σ
$\mathsf{codom}(f)$	42	The set $\bigcup_{\alpha \in \mathsf{dom}(f)} \mathsf{ftv}(f(\alpha))$
$d(\sigma)$	97	The degree of $P(\sigma)$
$\mathsf{dom}(Q)$	39	The set $\{\alpha_1, \ldots \alpha_n\}$
$\mathsf{dom}(\sigma)$	40	The domain of $\sigma/$
$\mathsf{dom}(f)$	42	The set $\{\alpha \mid f(\alpha) \neq \alpha\}$
$\mathsf{dom}_=(Q)$, $\mathsf{dom}_>(Q)$	207	Rigid domain of Q, flexible domain of Q
$\mathsf{dom}_1(C)$	73	The 1-level of context C
$\mathsf{dom}(C)$	71	The domain $\mathsf{dom}(\overline{C})$
$\mathsf{dom}(Q/I)$	104	See Definition 3.3.1
$\mathsf{dom}(Q/\sigma_1, \ldots \sigma_i)$	104	The domain $\mathsf{dom}(Q/\,\mathsf{ftv}(\sigma_1) \cup \ldots \mathsf{ftv}(\sigma_i))$
$\mathsf{dom}(Q/\sigma)$	104	The domain $\mathsf{dom}(Q/\,\mathsf{ftv}(\sigma))$
$\mathsf{fsub}\,(S)$	192	$\bigcup_{t \in S} \mathsf{fsub}\,(t)$
$\mathsf{fsub}\,(t)$	192	$\{t' \mid t \sqsubseteq_F t'\}$
$\mathsf{ftv}(\sigma)$	41	The set of free type variables of σ

continued. . .

Symbol	Page	Meaning
$\mathsf{ftv}(\sigma_1, \sigma_2)$	41	The set $\mathsf{ftv}(\sigma_1) \cup \mathsf{ftv}(\sigma_2)$
g or g^n	38	A type constructor, of arity n
I	101	An interface, that is, a set of type variables $\bar{\alpha}$
$\mathsf{level}(C)$	73	The level of context C
$\mathsf{nf}(\sigma)$	51	The normal form of σ
$P(\sigma)$	97	$w_X(\sigma)(X, X, X)$
$\mathsf{proj}(\sigma)$	40	The function mapping types to skeletons
Q	39	A prefix, *e.g.*(1)
$Q{\uparrow}\bar{\alpha}$	112	The splitting of Q according to $\bar{\alpha}$
$Q[\alpha]$	69	The representative of α in Q
$Q(\alpha)$	69	The bound of $Q[\alpha]$ in Q
$Q \Leftarrow (\alpha \diamond \sigma)$	123	Update algorithm, see figure 4.2
$Q \Leftarrow \alpha_1 \wedge \alpha_2$	124	Merge algorithm, see Definition 4.2.6
$\widehat{Q}$	52	The substitution extracted from Q
$\mathcal{R}^*$	34	The transitive closure of $\mathcal{R}$
$\mathcal{R}$	34	A meta-variable standing for a relation (depending on the context)
S	192	A set of System F types.
t	39	A skeleton
$\mathsf{utv}(\sigma)$	41	The set of unbound type variables of σ
ϑ	38	The set of type variables.
$\mathcal{V}$	51	The set $\{\sigma \mid \exists \alpha . \mathsf{nf}(\sigma) = \alpha\}$
$w_X(\sigma)$, $w_Y(\sigma)$, $w_Z(\sigma)$	91	Weights of σ (polynomials)
constructed form	55	See Definition 1.5.12
flexible binding	38	$(\alpha \geq \sigma)$
f invariant on $\bar{\alpha}$	42	$\bar{\alpha} \# \mathsf{dom}(f)$
rigid binding	38	$(\alpha = \sigma)$
unconstrained binding	38	$(\alpha \geq \bot)$ also written (α)